Scalable Semidefinite Programming*

Alp Yurtsever†, Joel A. Tropp‡, Olivier Fercoq§, Madeleine Udell¶, and Volkan Cevher†

Abstract. Semidefinite programming (SDP) is a powerful framework from convex optimization that has striking potential for data science applications. This paper develops a provably correct algorithm for solving large SDP problems by economizing on both the storage and the arithmetic costs. Numerical evidence shows that the method is effective for a range of applications, including relaxations of MaxCut, abstract phase retrieval, and quadratic assignment. Running on a laptop, the algorithm can handle SDP instances where the matrix variable has over 10^{13} entries.

Key words. Augmented Lagrangian, conditional gradient method, convex optimization, dimension reduction, first-order method, Frank–Wolfe method, MaxCut, phase retrieval, quadratic assignment, randomized linear algebra, semidefinite programming, sketching.

AMS subject classifications. Primary: 90C22, 65K05. Secondary: 65F99.

1. Motivation.

For a spectrum of challenges in data science, methodologies based on semidefinite programming offer remarkable performance both in theory and for small problem instances. Even so, practitioners often critique this approach by asserting that it is impossible to solve semidefinite programs (SDPs) at the scale demanded by real-world applications. We would like to argue against this article of conventional wisdom.

This paper proposes a new algorithm, called SketchyCGAL, that can solve very large SDPs to moderate accuracy. The algorithm marries a primal–dual optimization technique to a randomized sketch for low-rank matrix approximation. For every standard-form SDP that satisfies strong duality, SketchyCGAL provably finds a near-optimal low-rank approximation of a solution using limited storage and arithmetic. In contrast, given the same computational resources, other methods for large-scale SDP may fail where SketchyCGAL succeeds. In particular, SketchyCGAL needs dramatically less storage than Burer–Monteiro factorization [27, 22] for difficult problem instances [105].

In addition to the strong theoretical guarantees, we offer extensive evidence that SketchyCGAL is a practical optimization algorithm. For example, on a laptop, we can solve the

*Submitted to the editors 6 December 2019.

Funding: VC and AY have received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme under the grant agreement number 725594 (time-data) and the Swiss National Science Foundation (SNSF) under the grant number 200021_178865/1. JAT gratefully acknowledges ONR Awards N00014-11-1-0025, N00014-17-1-2146, and N00014-18-1-2363. MU gratefully acknowledges DARPA Award FA8750-17-2-0101.

†Laboratory for Information and Inference Systems, Department of Electrical Engineering, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland (alp.yurtsever@epfl.ch, volkan.cevher@epfl.ch, https://lions.epfl.ch/).

‡Department of Computing and Mathematical Sciences, California Institute of Technology, Pasadena, CA, USA (jtropp@cms.caltech.edu, http://users.cms.caltech.edu/~jtropp).

§Laboratoire Traitement et Communication d'Information, Télécom Paris, Institut Polytechnique de Paris, Palaiseau, France (olivier.fercoq@telecom-paris.fr, https://perso.telecom-paristech.fr/ofercoq/.)

¶Department of Operations Research and Information Engineering, Cornell University, Ithaca, NY, USA (udell@cornell.edu, https://people.orie.cornell.edu/mru8/).

MaxCut SDP for a sparse graph with almost 8 million vertices, where the matrix variable has about 10^{13} entries. We also tackle large phase retrieval problems arising from Fourier ptychography [55], as well as relaxations [117, 26] of the quadratic assignment problem.

We will first explain how our approach applies to the MaxCut SDP, and then we will enlarge our scope to include all standard-form SDPs.

1.1. Example: The maximum cut in a graph. To begin, we derive a fundamental SDP [37, 38, 47] that arises in combinatorial optimization. This example shows how SDPs arise from matrix optimization problems with rank constraints. It highlights why large SDPs are computationally challenging, and it allows us to illustrate the potential of our approach.

1.1.1. MaxCut. Consider an undirected graph $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ composed of the vertex set $\mathsf{V} = \{1, \dots, n\}$ and a set E that lists the m edges. The combinatorial Laplacian of the graph is the real positive-semidefinite (psd) matrix

$$\boldsymbol{L} := \sum\nolimits_{\{i,j\} \in \mathsf{E}} (\mathbf{e}_i - \mathbf{e}_j)(\mathbf{e}_i - \mathbf{e}_j)^* \in \mathbb{R}^{n \times n}, \tag{1.1}$$

where $\mathbf{e}_i \in \mathbb{R}^n$ denotes the ith standard basis vector and * refers to the (conjugate) transpose of a matrix or vector. The Laplacian $\boldsymbol{L}$ has at most $2m + n$ nonzero entries.

A *cut* is a subset S of the vertex set V, and the signed indicator of the cut is the vector

$$\boldsymbol{\chi}_{\mathsf{S}} \in \mathbb{R}^n \quad \text{where} \quad \boldsymbol{\chi}_{\mathsf{S}}(i) = \begin{cases} +1, & i \in \mathsf{S}; \\ -1, & i \in \mathsf{V} \setminus \mathsf{S}. \end{cases}$$

Define weight(S) to be the number of edges with one vertex in S and the other in $\mathsf{V} \setminus \mathsf{S}$. We can compute the weight of a cut algebraically: $4\,\mathrm{weight}(\mathsf{S}) = \boldsymbol{\chi}_{\mathsf{S}}{}^* \boldsymbol{L} \boldsymbol{\chi}_{\mathsf{S}}$. As a consequence, we can search for the maximum-weight cut in the graph via the discrete optimization problem

$$\text{maximize} \quad \boldsymbol{\chi}^* \boldsymbol{L} \boldsymbol{\chi} \quad \text{subject to} \quad \boldsymbol{\chi} \in \{\pm 1\}^n. \tag{1.2}$$

The formulation (1.2) is NP-hard [63]. One remedy is to relax it to an SDP.

Consider the matrix $\boldsymbol{X} = \boldsymbol{\chi}\boldsymbol{\chi}^*$ where $\boldsymbol{\chi} \in \{\pm 1\}^n$. The matrix $\boldsymbol{X}$ is psd; its diagonal entries equal one; and it has rank one. We can express the MaxCut problem (1.2) in terms of the matrix $\boldsymbol{X}$ by rewriting the objective as a trace. Bringing forward the implicit constraints on $\boldsymbol{X}$ and dropping the rank constraint, we arrive at the MaxCut SDP:

$$\text{maximize} \quad \operatorname{tr}(\boldsymbol{L}\boldsymbol{X}) \quad \text{subject to} \quad \operatorname{diag}(\boldsymbol{X}) = \mathbf{1}, \quad \boldsymbol{X} \text{ is psd.} \tag{1.3}$$

As usual, diag extracts the diagonal of a matrix, and $\mathbf{1} \in \mathbb{R}^n$ is the vector of ones.

The matrix solution $\boldsymbol{X}_\star$ of (1.3) does not immediately yield a cut. Let $\boldsymbol{x}_\star \boldsymbol{x}_\star^*$ be a best rank-one approximation of $\boldsymbol{X}_\star$ with respect to the Frobenius norm. Then the vector $\boldsymbol{\chi}_\star = \operatorname{sgn}(\boldsymbol{x}_\star)$ is the signed indicator of a cut $\mathsf{S}_\star$. In many cases, the cut $\mathsf{S}_\star$ yields an excellent solution to the MaxCut problem (1.2). We can also use $\boldsymbol{X}_\star$ to compute a cut that is provably near-optimal via a more involved randomized rounding procedure [47].

1.1.2. What's the issue? We specify an instance of the MaxCut SDP (1.3) by means of the Laplacian $\boldsymbol{L}$ of the graph, which has $O(m+n)$ nonzero entries. Our goal is to compute a best rank-one approximation of a solution to the SDP, which has $O(n)$ degrees of freedom. In other words, the total cost of representing the input and output of the problem is $O(m+n)$. Sadly, the matrix variable in (1.3) seems to require storage $O(n^2)$. For example, a graph G with two million vertices leads to an SDP (1.3) with almost two trillion real variables!

Storage is one of the main reasons that it has been challenging to solve large instances of the MaxCut SDP reliably. Undeterred, we raise a question:

> *Can we provably find a best rank-one approximation of a solution to the MaxCut SDP (1.3) with working storage $O(m+n)$? Can we achieve storage $O(n)$?*

We are not aware of any correct algorithm that can solve an arbitrary instance of (1.3) with a working storage guarantee better than $\Theta(\min\{m, n^{3/2}\})$; see section 8.

In addition to the limit on storage, a good algorithm should interact with the Laplacian $\boldsymbol{L}$ only through noninvasive, low-cost operations, such as matrix–vector multiplication.

1.1.3. A storage-optimal algorithm for the MaxCut SDP. Surprisingly, it is possible to achieve all the goals announced in the last paragraph.

Informal Theorem 1.1 (MaxCut via SketchyCGAL). *For any $\varepsilon, \zeta > 0$ and any Laplacian $\boldsymbol{L}$, the SketchyCGAL algorithm computes a $(1+\zeta)$-optimal rank-one approximation of an ε-optimal point of (1.3); see subsection 2.2. The working storage is $O(n/\zeta)$. The algorithm performs at most $\tilde{O}(\varepsilon^{-3})$ matrix–vector multiplies with the Laplacian $\boldsymbol{L}$, plus lower-order arithmetic. The algorithm is randomized; it succeeds with high probability over its random choices.*

Informal Theorem 1.1 follows from Theorem 6.3. As usual, $\tilde{O}$ suppresses constants and logarithmic factors. In our experience, SketchyCGAL works *better* than the theorem suggests: we find that SketchyCGAL succeeds using only $\tilde{O}(\varepsilon^{-5/4})$ matrix–vector multiplies with $\boldsymbol{L}$, provided the slightly larger storage budget $\tilde{O}((\varepsilon^{-1/4}+\zeta^{-1})n)$. See subsection 7.2 for numerical examples on sparse graphs with almost 10^7 vertices.

Remark 1.2 (Prior work). In contrast to Burer–Monteiro methods [105] and to the approximate complementarity paradigm [39], SketchyCGAL succeeds for every instance of MaxCut. At present, the fastest theoretical algorithm [69] for finding an ε-optimal solution to (1.3) requires $O(m)$ working storage and $\tilde{O}(\varepsilon^{-3.5})$ matrix–vector multiplies with $\boldsymbol{L}$.

1.2. A model problem. SketchyCGAL can solve all standard-form SDPs. To simplify some aspects of the presentation, we will focus on a model problem that includes an extra trace constraint. Appendix D explains how to extend SketchyCGAL to a more expressive problem template that includes standard-form SDPs with additional (conic) inequality constraints.

1.2.1. The trace-constrained SDP. We work over the field $\mathbb{F} = \mathbb{R}$ or $\mathbb{F} = \mathbb{C}$. For each $n \in \mathbb{N}$, define the set $\mathbb{S}_n := \mathbb{S}_n(\mathbb{F})$ of (conjugate) symmetric $n \times n$ matrices with entries in $\mathbb{F}$.

Introduce the set of psd matrices with trace one:

$$\boldsymbol{\Delta}_n := \{\boldsymbol{X} \in \mathbb{S}_n : \operatorname{tr} \boldsymbol{X} = 1 \text{ and } \boldsymbol{X} \text{ is psd}\}. \tag{1.4}$$

Our model problem is the following trace-constrained SDP:

$$\begin{aligned} \text{minimize} \quad & \operatorname{tr}(\boldsymbol{C}\boldsymbol{X}) \\ \text{subject to} \quad & \operatorname{tr}(\boldsymbol{A}_i\boldsymbol{X}) = b_i \quad \text{for } i = 1, \dots, d \quad \text{and} \quad \boldsymbol{X} \in \alpha\boldsymbol{\Delta}_n. \end{aligned} \tag{1.5}$$

The trace parameter $\alpha > 0$, each matrix $\boldsymbol{C}, \boldsymbol{A}_1, \dots, \boldsymbol{A}_d \in \mathbb{S}_n$, and $b_1, \dots, b_d \in \mathbb{R}$. We always assume that (1.5) satisfies strong duality with its standard-form dual problem.

Remark 1.3 (Standard-form SDPs). To solve a standard-form SDP, we replace the inclusion $\boldsymbol{X} \in \alpha\boldsymbol{\Delta}_n$ with the constraints that $\boldsymbol{X}$ is psd and $\operatorname{tr} \boldsymbol{X} \leq \alpha$ for a large enough parameter α.

1.2.2. Applications. The model problem (1.5) has diverse applications in statistics, signal processing, quantum information theory, combinatorics, and beyond. Evidently, the MaxCut SDP (1.3) is a special case of the model problem (1.5). The template (1.5) also covers alignment problems that appear in biological imaging [17] and in robotics [92]. It includes SDPs that arise from phase retrieval problems in imaging sciences [33, 104, 55] and in segmentation problems from computer vision [56]. It also supports contemporary machine learning tasks, such as certifying robustness of neural networks [90]. The possibilities are endless.

1.3. Complexity of SDP formulations and solutions. This section describes some special features that commonly appear in large, real-world SDPs. Our algorithm will exploit these features in an essential way, even as it provides guarantees for every instance of (1.5).

1.3.1. Structure of the problem data. The matrices $\boldsymbol{C}$ and $\boldsymbol{A}_i$ that appear in (1.5) are often highly structured, or sparse, or have low-rank. As such, we can specify the SDP using a small amount of information. In our work, we exploit this property by treating the problem data for the SDP (1.5) as a collection of black boxes that support specific linear algebraic operations. The algorithm for solving the SDP only needs to access the data via these black boxes, and we can make sure that these subroutines are implemented efficiently.

1.3.2. Low-rank solutions of SDPs. We will also capitalize on the fact that SDPs frequently have low-rank solutions, or the solutions are approximated well by low-rank matrices. There are several reasons why we can make this surmise.

Weakly-constrained SDPs. First, many SDPs have low-rank solutions just because they are weakly constrained. That is, the number d of linear inequalities in (1.5) is much smaller than the $\Theta(n^2)$ number of entries in the matrix variable. This situation often occurs in signal processing and statistics problems, where d reflects the amount of measured data.

Weakly constrained SDPs provably have low-rank solutions because of the geometry of the set of psd matrices; see [18, Prop. II(13.4) and Prob. II.14.5] and [87].

Fact 1.4 (Barvinok–Pataki). *When $\mathbb{F} = \mathbb{R}$, the SDP (1.5) has a solution with rank $r \leq \sqrt{2(d+1)}$. When $\mathbb{F} = \mathbb{C}$, there is a solution with rank $r \leq \sqrt{d+1}$.*

For example, the MaxCut SDP (1.3) admits a solution with rank $\sqrt{2(n+1)}$.

Although a weakly-constrained SDP can have solutions with high rank, a *generic* weakly-constrained SDP only admits a low-rank solution [5].

Fact 1.5 (Alizadeh et al.). *Let $\mathbb{F} = \mathbb{R}$. Except for a set of matrices $\{\boldsymbol{C}, \boldsymbol{A}_1, \dots, \boldsymbol{A}_d\}$ with measure zero, the solution set of the SDP (1.5) is a unique matrix with rank $r \leq \sqrt{2(d+1)}$.*

Our algorithm targets weakly constrained SDPs, but it works for all instances of (1.5).

Matrix rank minimization. Second, some SDPs are *designed* to produce a low-rank matrix that satisfies a system of linear matrix equations. This idea goes back to the control literature [76, 86], and it was explored thoroughly in Fazel's thesis [40]. Early applications include Euclidean distance matrix completion [2] and collaborative filtering [96]. Extensive empirical work indicates that these SDPs often produce low-rank solutions.

Structural properties. There are other reasons that an SDP must have a low-rank solution. For instance, consider the optimal power flow SDPs developed by Lavaei and Low [68], where the rank of the solution is controlled by the geometry of the power grid.

1.3.3. Algorithms? To summarize, many realistic SDPs have structured data, and they admit solutions that are (close to) low rank. Are there algorithms that can exploit these features? Although there are a number of methods that attempt to do so, none can provably solve every SDP while controlling storage and arithmetic costs. See section 8 for related work.

1.3.4. Finding low-rank solutions? Why has it been so difficult to develop provably correct algorithms for finding low-rank solutions to structured SDPs? Most approaches that try to control the rank run headlong into a computational complexity barrier: For any fixed rank parameter r, it is NP-hard to solve the model problem (1.5) if the variable $\boldsymbol{X}$ is also constrained to be a rank-r matrix [40, p. 7].

To escape this sticky fact, we revise the computational goal, following [115]. The key insight is to seek a rank-r matrix that *approximates a solution* to (1.5). See subsection 2.2 for a detailed explanation. This shift in perspective opens up new algorithmic prospects.

1.4. Contributions. This paper explains how to harness the favorable properties that are common in large SDPs. The SketchyCGAL algorithm solves the SDP using a primal–dual optimization method [110] developed by a subset of the authors. Instead of storing the psd matrix variable, we maintain a compressed representation by means of a matrix sketching technique [100]. After the optimization algorithm terminates, we extract from the sketch a low-rank approximation of the solution of the SDP. This idea leads to a practical, provably correct SDP solver that economizes on storage and arithmetic.

Informal Theorem 1.6 (The model problem via SketchyCGAL). *Assume that the model problem (1.5) satisfies strong duality. For any $\varepsilon, \zeta > 0$ and any rank parameter r, the SketchyCGAL algorithm computes a $(1+\zeta)$-optimal rank-r approximation of an ε-optimal point of (1.5); see subsection 2.2. The storage cost is $O(d + rn/\zeta)$. Most of the arithmetic consists of $\tilde{O}(\varepsilon^{-3})$ matrix–vector products with each matrix $\boldsymbol{C}, \boldsymbol{A}_1, \ldots, \boldsymbol{A}_d$ from the problem data. The algorithm succeeds with high probability.*

Theorem 6.3 contains full theoretical details. Note that the storage $O(d + rn)$ is the minimum for any primal–dual algorithm that returns a rank-r solution to (1.5). The arithmetic requirements can be reduced at the expense of additional storage. Computational evidence indicates that the true arithmetic costs are much lower than we can prove; see section 7. We believe SketchyCGAL is the first SDP solver with these guarantees. And it actually works!

1.5. Roadmap. Section 2 presents an abstract framework for studying SDPs that highlights the challenges associated with large problems. Section 3 outlines a primal–dual algorithm, called CGAL, for solving the model problem (1.5). Sections 4 and 5 introduce some

methods from randomized linear algebra that we use to control storage and arithmetic costs. Section 6 develops the SketchyCGAL algorithm, its convergence theory, and its resource usage guarantees. Appendix D describes some extensions of the model problem. We present a numerical study of SketchyCGAL in section 7. Section 8 discusses related work.

1.6. Notation. The symbol $\|\cdot\|_{\mathrm{F}}$ denotes the Frobenius norm, while $\|\cdot\|_*$ is the nuclear norm (i.e., Schatten-1). The unadorned norm $\|\cdot\|$ refers to the ℓ_2 norm of a vector, the spectral norm of a matrix, or the operator norm of a linear map from $(\mathbb{S}_n, \|\cdot\|_{\mathrm{F}})$ to $(\mathbb{R}^d, \|\cdot\|)$. We write $\langle\cdot,\,\cdot\rangle$ for both the ℓ_2 inner product on vectors and the trace inner product on matrices.

The map $[\![\boldsymbol{M}]\!]_r$ returns an r-truncated singular-value decomposition of the matrix $\boldsymbol{M}$, which is a best rank-r approximation with respect to every unitarily invariant norm [77].

We use the standard computer science interpretation of the asymptotic notation $O, \tilde{O}, \Theta$.

2. Scalable semidefinite programming.

To solve the model problem (1.5) efficiently, we need to exploit structure inherent in the problem data. This section outlines an abstract approach that directs our attention to the core computational difficulties.

2.1. Abstract form of the model problem. Let us instate compact notation for the linear constraints in the model problem (1.5). Define a linear map $\mathcal{A}$ and its adjoint $\mathcal{A}^*$ via

$$\begin{aligned} \mathcal{A} &: \mathbb{S}_n \to \mathbb{R}^d \quad \text{where} \quad \mathcal{A}\boldsymbol{X} = \big[\langle \boldsymbol{A}_1,\, \boldsymbol{X}\rangle \quad \dots \quad \langle \boldsymbol{A}_d,\, \boldsymbol{X}\rangle\big]; \\ \mathcal{A}^* &: \mathbb{R}^d \to \mathbb{S}_n \quad \text{where} \quad \mathcal{A}^*\boldsymbol{z} = \sum\nolimits_{i=1}^{d} z_i \boldsymbol{A}_i. \end{aligned} \tag{2.1}$$

We equip the linear map with the operator norm $\|\mathcal{A}\| := \|\mathcal{A}\|_{\mathrm{F}\to\ell_2}$. Form the vector $\boldsymbol{b} := (b_1, \dots, b_d) \in \mathbb{R}^d$ of constraint values. In this notation, (1.5) becomes

$$\text{minimize} \quad \langle \boldsymbol{C},\, \boldsymbol{X}\rangle \quad \text{subject to} \quad \mathcal{A}\boldsymbol{X} = \boldsymbol{b}, \quad \boldsymbol{X} \in \alpha\boldsymbol{\Delta}_n. \tag{2.2}$$

Problem instances are parameterized by the tuple $(\boldsymbol{C}, \mathcal{A}, \boldsymbol{b}, \alpha)$.

2.2. Approximate solutions. Let $\boldsymbol{X}_\star$ be a solution to the model problem (2.2). For $\varepsilon \geq 0$, we say that a matrix $\boldsymbol{X}$ is *ε-optimal* for (2.2) when

$$\|\mathcal{A}\boldsymbol{X} - \boldsymbol{b}\| \leq \varepsilon \quad \text{and} \quad \langle \boldsymbol{C},\, \boldsymbol{X}\rangle - \langle \boldsymbol{C},\, \boldsymbol{X}_\star\rangle \leq \varepsilon.$$

Many optimization algorithms aim to produce ε-optimal points.

As we saw in subsection 1.3.2, there are many situations where the solutions to (2.2) have low rank, or they admit accurate low-rank approximations. The ε-optimal points inherit these properties for sufficiently small ε. This insight suggests a new computational goal.

For a rank parameter r, we will seek a rank-r matrix $\widehat{\boldsymbol{X}}$ that approximates an ε-optimal point $\boldsymbol{X}$. More precisely, for a fixed suboptimality parameter $\zeta > 0$, we want

$$\|\boldsymbol{X} - \widehat{\boldsymbol{X}}\|_* \leq (1+\zeta)\cdot\|\boldsymbol{X} - [\![\boldsymbol{X}]\!]_r\|_* \quad \text{where } \operatorname{rank}\widehat{\boldsymbol{X}} \leq r \text{ and } \boldsymbol{X} \text{ is } \varepsilon\text{-optimal}. \tag{2.3}$$

Given (2.3), if the ε-optimal point $\boldsymbol{X}$ is close to *any* rank-r matrix, then the rank-r approximate solution $\widehat{\boldsymbol{X}}$ is also close to the ε-optimal point $\boldsymbol{X}$. This formulation is advantageous because it is easier to compute and to store the low-rank matrix $\widehat{\boldsymbol{X}}$.

2.3. Black-box presentation of problem data. To develop scalable algorithms for (2.2), it is productive to hide the internal complexity of the problem instance from the algorithm [115]. To do so, we treat $\boldsymbol{C}$ and $\mathcal{A}$ as black boxes that support three primitive computations:

$$\text{(2.4)} \qquad \boxed{\begin{array}{c}\text{❶} \quad \boldsymbol{u} \mapsto \boldsymbol{C}\boldsymbol{u} \\ \mathbb{R}^n \to \mathbb{R}^n\end{array}} \qquad \boxed{\begin{array}{c}\text{❷} \quad (\boldsymbol{u}, \boldsymbol{z}) \mapsto (\mathcal{A}^*\boldsymbol{z})\boldsymbol{u} \\ \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}^n\end{array}} \qquad \boxed{\begin{array}{c}\text{❸} \quad \boldsymbol{u} \mapsto \mathcal{A}(\boldsymbol{u}\boldsymbol{u}^*) \\ \mathbb{R}^n \to \mathbb{R}^d\end{array}}$$

The vectors $\boldsymbol{u} \in \mathbb{R}^n$ and $\boldsymbol{z} \in \mathbb{R}^d$ are arbitrary. Although these functions may seem abstract, they are often quite natural and easy to implement well. For example, see subsection 2.5.

We will formulate algorithms for (2.2) that interact with the problem data only through the operations (2.4). We tacitly assume that the primitives require minimal storage and arithmetic; otherwise, it may be impossible to develop a truly efficient algorithm.

Remark 2.1 (Inexact oracles). We assume the primitives (2.4) are implemented exactly. An interesting future direction is to study approximate or stochastic versions.

2.4. Resource usage. Our goal is to develop a scalable algorithm for the template (2.2) by computing a rank-r approximate solution that satisfies (2.3). Running time is less of an issue than storage because modern computers have fast processors but limited memory.

So what can we hope to achieve in terms of storage? First, observe that we need to specify the constraint vector $\boldsymbol{b} \in \mathbb{R}^d$ to determine an arbitrary instance of the model problem (2.2). Second, observe that a rank-r matrix with dimension n has $\Theta(rn)$ degrees of freedom. Therefore, we must be prepared to spend $\Theta(d + rn)$ numbers just to express the input and output of an SDP solver. The working storage of SketchyCGAL achieves this bound. This cost should be viewed in contrast to the $\Theta(n^2)$ degrees of freedom in the matrix variable in (2.2).

We tabulate the arithmetic costs of an algorithm by counting the number of times we apply each primitive, plus the total amount of extra arithmetic. For lower bounds on the arithmetic operations needed to solve an SDP to moderate accuracy, see [44, Thm. 2].

Remark 2.2 (Communication). Data transfer is the dominant expense in most large-scale applications. We do not control communication costs directly, but limiting the storage and arithmetic has the ancillary benefit of reducing data transfer.

2.5. Example: The MaxCut SDP. To provide a concrete example, let us summarize the meaning of the primitives and the desired storage costs for solving the MaxCut SDP (1.3).

- The primitive ❶ : $\boldsymbol{u} \mapsto -\boldsymbol{L}\boldsymbol{u}$. In the typical case that the Laplacian $\boldsymbol{L}$ is sparse, this amounts to a sparse matrix–vector multiply.
- The primitive ❷ : $(\boldsymbol{u}, \boldsymbol{z}) \mapsto (\operatorname{diag}^* \boldsymbol{z})\boldsymbol{u} = (z_1 u_1, \dots, z_n u_n)$.
- The primitive ❸ : $\boldsymbol{u} \mapsto \operatorname{diag}(\boldsymbol{u}\boldsymbol{u}^*) = (|u_1|^2, \dots, |u_n|^2)$.
- The MaxCut SDP has n linear constraints, and we seek a rank-one approximation of the solution. Thus, we desire an algorithm that operates with $\Theta(n)$ working storage.

Note that we still need $\Theta(m)$ numbers to store the Laplacian $\boldsymbol{L}$ of a generic graph with m edges. But we do not charge the optimization algorithm for this storage because the algorithm only interacts with $\boldsymbol{L}$ through the primitives (2.4).

3. An algorithm for the model problem. We will develop a scalable method for the model problem (2.2) by enhancing an existing algorithm, called CGAL [110], developed by a subset of the authors. This method works well but lacks strong storage and arithmetic guarantees. This section summarizes the CGAL method and its convergence properties. Subsequent sections introduce additional ideas that we need to control resource usage, culminating with the SketchyCGAL algorithm in section 6.

3.1. The augmented problem. For a parameter $\beta > 0$, we revise the problem (2.2) by introducing an extra term in the objective:

$$\text{minimize} \quad \langle \boldsymbol{C}, \boldsymbol{X} \rangle + \frac{\beta}{2}\|\mathcal{A}\boldsymbol{X} - \boldsymbol{b}\|^2 \quad \text{subject to} \quad \mathcal{A}\boldsymbol{X} = \boldsymbol{b}, \quad \boldsymbol{X} \in \alpha\boldsymbol{\Delta}_n. \tag{3.1}$$

The original problem (2.2) and the augmented problem (3.1) share the same optimal value and optimal set. But the new formulation has several benefits: the augmented objective cooperates with the affine constraint to penalize infeasible points, and the dual of the augmented problem is smoother than the dual of the original problem. See [20, Ch. 2] for background.

3.2. The augmented Lagrangian. We construct the Lagrangian L_β of the augmented problem (3.1) by introducing a dual variable $\boldsymbol{y} \in \mathbb{R}^d$ and promoting the affine constraint:

$$L_\beta(\boldsymbol{X}; \boldsymbol{y}) := \langle \boldsymbol{C}, \boldsymbol{X} \rangle + \langle \boldsymbol{y}, \mathcal{A}\boldsymbol{X} - \boldsymbol{b} \rangle + \frac{\beta}{2}\|\mathcal{A}\boldsymbol{X} - \boldsymbol{b}\|^2 \quad \text{for } \boldsymbol{X} \in \alpha\boldsymbol{\Delta}_n \text{ and } \boldsymbol{y} \in \mathbb{R}^d. \tag{3.2}$$

For reference, note that the partial derivatives of the augmented Lagrangian L_β satisfy

$$\begin{aligned} \partial_{\boldsymbol{X}} L_\beta(\boldsymbol{X}; \boldsymbol{y}) &= \boldsymbol{C} + \mathcal{A}^*\boldsymbol{y} + \beta\mathcal{A}^*(\mathcal{A}\boldsymbol{X} - \boldsymbol{b}) \\ \partial_{\boldsymbol{y}} L_\beta(\boldsymbol{X}; \boldsymbol{y}) &= \mathcal{A}\boldsymbol{X} - \boldsymbol{b}. \end{aligned} \tag{3.3}$$

We attempt to minimize the augmented Lagrangian L_β with respect to the primal variable $\boldsymbol{X}$, while we attempt to maximize with respect to the dual variable $\boldsymbol{y}$.

3.3. The augmented Lagrangian strategy. The form of the augmented Lagrangian suggests an algorithm. We generate a sequence $\{(\boldsymbol{X}_t; \boldsymbol{y}_t)\}$ of primal–dual pairs by alternately minimizing over the primal variable $\boldsymbol{X}$ and taking a gradient step in the dual variable $\boldsymbol{y}$.

1. **Initialize:** Choose $\boldsymbol{X}_0 \in \alpha\boldsymbol{\Delta}_n$ and $\boldsymbol{y}_0 \in \mathbb{R}^d$.
2. **Primal step:** $\boldsymbol{X}_{t+1} \in \arg\min\{L_\beta(\boldsymbol{X}; \boldsymbol{y}_t) : \boldsymbol{X} \in \alpha\boldsymbol{\Delta}_n\}$.
3. **Dual step:** $\boldsymbol{y}_{t+1} = \boldsymbol{y}_t + \beta(\mathcal{A}\boldsymbol{X} - \boldsymbol{b})$.

As we proceed, we can also increase the smoothing parameter β to make violations of the affine constraint in (3.1) more and more intolerable.

The augmented Lagrangian strategy is powerful, but it is hard to apply in this setting because of the cost of implementing the primal step, even approximately.

3.4. The CGAL iteration. The CGAL iteration [110] is related to the augmented Lagrangian (AL) paradigm, but the primal steps are inspired by the conditional gradient method (CGM). CGAL identifies an update direction for the primal variable by minimizing a linear proxy for the augmented Lagrangian. We take a small primal step in the update direction,

and we improve the dual variable by taking a small gradient step. At each iteration, the smoothing parameter increases according to a fixed schedule.

This subsection outlines the steps in the CGAL iteration. Afterward, we give *a priori* guarantees on the convergence rate. Pseudocode for CGAL appears in Algorithm A.1.

3.4.1. Initialization. Let $\beta_0 > 0$ be an initial smoothing parameter, and fix a (large) bound $K > 0$ on the maximum allowable size of the dual variable. We also assume that we have access to the norm $\|\mathcal{A}\|$ of the constraint matrix, or—failing that—a *lower* bound.

Begin with an arbitrary choice $\boldsymbol{X}_1 \in \mathbb{S}_n$ for the primal matrix variable, and set the initial dual vector variable $\boldsymbol{y}_1 = \mathbf{0}$.

3.4.2. Primal updates via linear minimization. At iteration $t = 1, 2, 3, \dots$, we increase the smoothing parameter to $\beta_t := \beta_0\sqrt{t+1}$, and we form the partial derivative of the augmented Lagrangian with respect to the primal variable at the current pair of iterates:

$$\boldsymbol{D}_t := \partial_{\boldsymbol{X}} L_{\beta_t}(\boldsymbol{X}_t; \boldsymbol{y}_t) = \boldsymbol{C} + \mathcal{A}^*(\boldsymbol{y}_t + \beta_t(\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b})). \tag{3.4}$$

Then we compute an update $\boldsymbol{H}_t \in \alpha\boldsymbol{\Delta}_t$ by finding the feasible point that is most correlated with the negative gradient $-\boldsymbol{D}_t$. This amounts to a linear minimization problem:

$$\begin{aligned} \boldsymbol{H}_t &\in \arg\min\{\langle \boldsymbol{D}_t, \boldsymbol{H}\rangle : \boldsymbol{H} \in \alpha\boldsymbol{\Delta}_n\} \\ &= \operatorname{convex\,hull}\{\alpha\boldsymbol{v}\boldsymbol{v}^* : \boldsymbol{v} \text{ is a unit-norm minimum eigenvector of } \boldsymbol{D}_t\}. \end{aligned} \tag{3.5}$$

In particular, we may take $\boldsymbol{H}_t = \alpha\boldsymbol{v}_t\boldsymbol{v}_t^*$ for a minimum eigenvector $\boldsymbol{v}_t$ of the gradient $\boldsymbol{D}_t$. Next, update the matrix primal variable by taking a small step in the direction $\boldsymbol{H}_t$:

$$\boldsymbol{X}_{t+1} := (1-\eta_t)\boldsymbol{X}_t + \eta_t\boldsymbol{H}_t \in \alpha\boldsymbol{\Delta}_n \quad \text{where} \quad \eta_t := \frac{2}{t+1}. \tag{3.6}$$

The appeal of this approach is that it only requires a single eigenvector computation (3.5). Moreover, we need not compute the eigenvector accurately; see subsection 3.4.4 for details.

3.4.3. Dual updates via gradient ascent. Next, we update the dual variable by taking a small gradient step on the dual variable in the augmented Lagrangian:

$$\boldsymbol{y}_{t+1} = \boldsymbol{y}_t + \gamma_t\,\partial_{\boldsymbol{y}} L_{\beta_t}(\boldsymbol{X}_{t+1}; \boldsymbol{y}_t) = \boldsymbol{y}_t + \gamma_t(\mathcal{A}\boldsymbol{X}_{t+1} - \boldsymbol{b}). \tag{3.7}$$

The dual step size γ_t is the largest number that satisfies the conditions

$$\gamma_t\|\mathcal{A}\boldsymbol{X}_{t+1} - \boldsymbol{b}\|^2 \le \frac{4\alpha^2\beta_0}{(t+1)^{3/2}}\|\mathcal{A}\|^2 \quad \text{and} \quad 0 \le \gamma_t \le \beta_0. \tag{3.8}$$

We omit the dual step if it makes the dual variable too large. More precisely, if (3.7) and (3.8) result in $\|\boldsymbol{y}_{t+1}\| > K$, then we set $\boldsymbol{y}_{t+1} = \boldsymbol{y}_t$ instead. This is the CGAL iteration.

3.4.4. Approximate eigenvector computations. It is crucial that the CGAL strategy provably works if we replace (3.5) with an approximate minimum eigenvector computation. At step t, suppose that we find an update direction

$$\boldsymbol{H}_t := \alpha\boldsymbol{v}_t\boldsymbol{v}_t^* \quad \text{where} \quad \boldsymbol{v}_t^*\boldsymbol{D}_t\boldsymbol{v}_t \le \lambda_{\min}(\boldsymbol{D}_t) + \frac{1}{\sqrt{t+1}}\|\boldsymbol{D}_t\| \tag{3.9}$$

for a unit vector $\boldsymbol{v}_t$. As usual, $\lambda_{\min}$ is the minimum eigenvalue. The matrix $\boldsymbol{H}_t$ from (3.9) serves in place of a solution to (3.5). We explain how to solve the subproblem (3.9) in section 4.

3.4.5. Solution quality. Given a primal–dual pair $(\boldsymbol{X}_t; \boldsymbol{y}_t)$, we can assess the quality of the primal solution to the model problem (2.2) and decide when to halt. See Appendix A.4.

3.5. Convergence guarantees for CGAL. The following result describes the convergence of the CGAL iteration. The analysis is adapted from [110, Thm. 3.1]; see Appendix A for details and a recapitulation of the proof.

Fact 3.1 (CGAL: Convergence). *Assume problem (2.2) satisfies strong duality. The CGAL iteration (subsection 3.4) with approximate eigenvector computations (3.9) yields a sequence* $\{\boldsymbol{X}_t : t = 1, 2, 3, \dots\} \subset \alpha\boldsymbol{\Delta}_n$ *that satisfies*

$$\|\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}\| \leq \frac{\mathit{Const}}{\sqrt{t}} \quad \text{and} \quad |\langle \boldsymbol{C}, \boldsymbol{X}_t\rangle - \langle \boldsymbol{C}, \boldsymbol{X}_\star\rangle| \leq \frac{\mathit{Const}}{\sqrt{t}}. \tag{3.10}$$

The matrix $\boldsymbol{X}_\star$ *solves (2.2). The constant depends on the problem data* $(\boldsymbol{C}, \mathcal{A}, \boldsymbol{b}, \alpha)$*, the minimum Euclidean norm of a dual solution, and the algorithm parameters* β_0 *and* K*,*

In view of (3.10), CGAL produces a primal iterate $\boldsymbol{X}_T$ that is ε-optimal after $T = O(\varepsilon^{-2})$ iterations. The numerical work in [110, Sec. 5] shows that CGAL usually achieves an ε-optimal point after $T = O(\varepsilon^{-1})$ iterations.

The analysis behind Fact 3.1 indicates that CGAL converges more quickly if we precondition the problem data by rescaling; see subsection 7.1.2. This step is critical in practice.

3.6. Distributed computation. Since CGAL builds on the augmented Lagrangian framework, we can apply it even when the problem is too large to solve on one computational node. In particular, when d is large, it may be advantageous to partition the constraint matrices $\boldsymbol{A}_i$ and the associated dual variables y_i among several workers. Distributed CGAL has a structure similar to the alternating directions method of multipliers (ADMM) [24].

4. Approximate eigenvectors. Most of the computation in CGAL occurs in the approximate eigenvector step (3.9). Here, we encounter a tradeoff between arithmetic and storage.

4.1. Krylov methods. The approximate minimum eigenvector computation (3.9) involves a matrix of the form $\boldsymbol{D}_t = \boldsymbol{C} + \mathcal{A}^*\boldsymbol{w}_t$. Observe that we can multiply the matrix $\boldsymbol{D}_t$ by any vector using the primitives (2.4)❶❷. Krylov methods compute eigenvectors of $\boldsymbol{D}_t$ by repeated matrix–vector multiplication with $\boldsymbol{D}_t$, so they are obvious tools for the subproblem (3.9).

Unfortunately, CGAL tends to generate matrices $\boldsymbol{D}_t$ that have clustered eigenvalues. These matrices challenge standard eigenvector software, such as ARPACK [70], the engine behind the MATLAB command `eigs`. Instead, we retreat to more classical techniques.

4.2. A randomized shifted power method. First, let us consider the situation where storage is the overriding concern. In this case, we run the shifted power method with a random starting vector; see Algorithm 4.1. Kuczyński & Woźniakowski [65, Thm. 4.1(a)] have obtained error bounds for this type of algorithm.

Fact 4.1 (Randomized shifted power method). *Let* $\boldsymbol{M} \in \mathbb{S}_n$*. For* $\varepsilon \in (0, 1]$ *and* $\delta \in (0, 1]$*, the shifted power method, Algorithm 4.1, computes a unit vector* $\boldsymbol{u} \in \mathbb{F}^n$ *that satisfies*

$$\boldsymbol{u}^*\boldsymbol{M}\boldsymbol{u} \leq \lambda_{\min}(\boldsymbol{M}) + \frac{3\varepsilon}{2}\|\boldsymbol{M}\| \quad \textit{with probability at least } 1 - \delta$$

Algorithm 4.1 ApproxMinEvec via randomized shifted power method (subsection 4.2).

Input: Input matrix $\boldsymbol{M} \in \mathbb{S}_n$ and maximum number q of iterations
Output: Approximate minimum eigenpair $(\xi, \boldsymbol{v}) \in \mathbb{R} \times \mathbb{F}^n$ of the matrix $\boldsymbol{M}$

```
function ApproxMinEvec(M; q)
    σ ← 2‖M‖                                  ▷ Use normest to estimate shift
    v ← randn(n, 1)/√n                        ▷ Random initial vector
    for i ← 1, 2, 3, ..., q do
        v ← σv − Mv                           ▷ Power method for σI − M
        v ← v/‖v‖                             ▷ Approx. minimum eigenvector of M
    ξ ← v*(Mv)                                ▷ Approx. minimum eigenvalue of M
```

after $q \geq \frac{1}{2} + \varepsilon^{-1} \log(n/\delta^2)$ *iterations.*[1] *The arithmetic cost is* $O(q)$ *matrix–vector multiplies with* $\boldsymbol{M}$ *and* $O(qn)$ *extra operations. The working storage is about* $2n$ *numbers.*

With constant probability, we solve the eigenvector problem (3.9) successfully in every iteration t of CGAL if we use $q_t = O(t^{1/2} \log(tn))$ iterations of Algorithm 4.1. In practice, we implement CGAL with the concrete choice $q_t = 8t^{1/2} \log n$, which behaves well empirically.

4.3. A randomized Lanczos method. Suppose we are willing to allocate a moderate amount of storage to perform the approximate eigenvector computation. In this case, we can run the Lanczos iteration with a random starting vector [48, Sec. 10.1]. See Algorithm 4.2 for pseudocode. Kuczyński & Woźniakowski [65, Thm. 4.2(a)] have obtained error bounds.

Fact 4.2 (Randomized Lanczos method). *Let* $\boldsymbol{M} \in \mathbb{S}_n$. *For* $\varepsilon \in (0, 1]$ *and* $\delta \in (0, 0.5]$, *the randomized Lanczos method, Algorithm 4.2, computes a unit vector* $\boldsymbol{u} \in \mathbb{F}^n$ *that satisfies*

$$\boldsymbol{u}^* \boldsymbol{M} \boldsymbol{u} \leq \lambda_{\min}(\boldsymbol{M}) + \frac{\varepsilon}{8} \|\boldsymbol{M}\| \quad \text{with probability at least } 1 - 2\delta$$

after $q \geq \frac{1}{2} + \varepsilon^{-1/2} \log(n/\delta^2)$ *iterations. The arithmetic cost is at most* q *matrix–vector multiplies with* $\boldsymbol{M}$ *and* $O(qn + q^2)$ *extra operations. The working storage is* $O(qn)$.

With constant probability, we solve the eigenvector problem (3.9) successfully in every iteration t of CGAL if we use $q_t = O(t^{1/4} \log(tn))$ iterations of Algorithm 4.2. In practice, we implement CGAL with the explicit choice $q_t = t^{1/4} \log n$.

5. Sketching and reconstruction of a psd matrix. The CGAL iteration generates a psd matrix that solves the model problem (2.2) via a sequence (3.6) of rank-one linear updates. To control storage costs, SketchyCGAL will retain only a summary of the psd matrix variable. This section outlines the *Nyström sketch*, an established method [49, 46, 72, 100] that can track the evolving psd matrix and then report a provably accurate low-rank approximation.

5.1. Sketching and updates. Consider a psd input matrix $\boldsymbol{X} \in \mathbb{S}_n$. Let R be a parameter that modulates the storage cost of the sketch and the quality of the matrix approximation.

[1] All logarithms are base-e.

Algorithm 4.2 ApproxMinEvec via randomized Lanczos method (subsection 4.3).

Input: Input matrix $\boldsymbol{M} \in \mathbb{S}_n$ and maximum number q of iterations
Output: Approximate minimum eigenpair $(\xi, \boldsymbol{v}) \in \mathbb{R} \times \mathbb{F}^n$ of the matrix $\boldsymbol{M}$

function ApproxMinEvec($\boldsymbol{M}$; q)
  $\boldsymbol{v}_1 \leftarrow \texttt{randn}(n, 1)$ ▷ Aggregate $\boldsymbol{v}_i$ as columns of $\boldsymbol{V}$
  $\boldsymbol{v}_1 \leftarrow \boldsymbol{v}_1 / \|\boldsymbol{v}_1\|$
  for $i \leftarrow 1, 2, 3, \ldots, \min\{q, n-1\}$ **do**
    $\alpha_i \leftarrow \mathrm{Re}(\boldsymbol{v}_i^*(\boldsymbol{M}\boldsymbol{v}_i))$
    $\boldsymbol{v}_{i+1} \leftarrow \boldsymbol{M}\boldsymbol{v}_i - \alpha_i \boldsymbol{v}_i - \beta_{i-1}\boldsymbol{v}_{i-1}$ ▷ Lanczos iteration; $\beta_0 \boldsymbol{v}_0 = \boldsymbol{0}$
    if $\beta_i = 0$ **then break** ▷ Found an invariant subspace!
    $\boldsymbol{v}_{i+1} \leftarrow \boldsymbol{v}_{i+1} / \beta_i$
  $\boldsymbol{T} \leftarrow \texttt{tridiag}(\boldsymbol{\beta}_{1:(i-1)}, \boldsymbol{\alpha}_{1:i}, \boldsymbol{\beta}_{1:(i-1)})$ ▷ Form tridiagonal matrix
  $[\boldsymbol{U}, \boldsymbol{D}] \leftarrow \texttt{eig}(\boldsymbol{T})$
  $[\xi, \texttt{ind}] \leftarrow \texttt{min}(\texttt{diag}(\boldsymbol{D}))$ ▷ Approx. minimum eigenvalue
  $\boldsymbol{v} \leftarrow \boldsymbol{V}_{1:i}\boldsymbol{U}(:, \texttt{ind}(1))$ ▷ Approx. minimum eigenvector

To construct the Nyström sketch, we draw and fix a standard normal[2] test matrix $\boldsymbol{\Omega} \in \mathbb{F}^{n \times R}$. Our summary, or *sketch*, of the matrix $\boldsymbol{X}$ takes the form

$$\boldsymbol{S} = \boldsymbol{X}\boldsymbol{\Omega} \in \mathbb{F}^{n \times R}. \tag{5.1}$$

The sketch supports linear rank-one updates to $\boldsymbol{X}$. Indeed, we can track the evolution

$$\begin{aligned} \boldsymbol{X} &\leftarrow (1-\eta)\,\boldsymbol{X} + \eta\,\boldsymbol{v}\boldsymbol{v}^* \qquad \text{for } \eta \in [0,1] \text{ and } \boldsymbol{v} \in \mathbb{F}^n \\ \text{via} \quad \boldsymbol{S} &\leftarrow (1-\eta)\,\boldsymbol{S} + \eta\,\boldsymbol{v}(\boldsymbol{v}^*\boldsymbol{\Omega}). \end{aligned} \tag{5.2}$$

The test matrix $\boldsymbol{\Omega}$ and the sketch $\boldsymbol{S}$ require storage of $2Rn$ numbers in $\mathbb{F}$. The arithmetic cost of the linear update (5.2) to the sketch is $\Theta(Rn)$ numerical operations.

Remark 5.1 (Structured random matrices). We can reduce storage costs by a factor of two by using a structured random matrix in place of $\boldsymbol{\Omega}$. For example, see [102, Sec. 3] or [98].

5.2. The reconstruction process. Given the test matrix $\boldsymbol{\Omega}$ and the sketch $\boldsymbol{S} = \boldsymbol{X}\boldsymbol{\Omega}$, we form a rank-$R$ approximation $\widehat{\boldsymbol{X}}$ of the sketched matrix $\boldsymbol{X}$. The approximation is defined by

$$\widehat{\boldsymbol{X}} := \boldsymbol{S}(\boldsymbol{\Omega}^*\boldsymbol{S})^\dagger \boldsymbol{S}^* = (\boldsymbol{X}\boldsymbol{\Omega})(\boldsymbol{\Omega}^*\boldsymbol{X}\boldsymbol{\Omega})^\dagger(\boldsymbol{X}\boldsymbol{\Omega})^*, \tag{5.3}$$

where † is the pseudoinverse. This reconstruction is called a *Nyström approximation*. We often truncate $\widehat{\boldsymbol{X}}$ by replacing it with its best rank-r approximation $[\![\widehat{\boldsymbol{X}}]\!]_r$ for some $r \leq R$.

See Algorithm 5.1 for a numerically stable implementation of the Nyström approximation (5.3). The algorithm takes $\Theta(R^2 n)$ numerical operations and $\Theta(Rn)$ storage.

[2]Each entry of the matrix is an independent Gaussian random variable with mean zero and variance one. In the complex setting, the real and imaginary parts of each entry are independent standard normal variables.

Algorithm 5.1 NystromSketch implementation (see section 5)

Input: Dimension n of input matrix, size R of sketch
Output: Rank-R approximation $\widehat{\boldsymbol{X}}$ of sketched matrix in factored form $\widehat{\boldsymbol{X}} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^*$, where $\boldsymbol{U} \in \mathbb{F}^{n\times R}$ has orthonormal columns and $\boldsymbol{\Lambda} \in \mathbb{R}^{R\times R}$ is nonnegative diagonal
Recommendation: Choose R as large as possible, given storage and arithmetic constraints

function NystromSketch.Init(n, R)
  $\boldsymbol{\Omega} \leftarrow \texttt{randn}(n, R)$ ▷ Draw and fix random test matrix
  $\boldsymbol{S} \leftarrow \texttt{zeros}(n, R)$ ▷ Form sketch of zero matrix

function NystromSketch.RankOneUpdate($\boldsymbol{v}$, η) ▷ Implements (5.2)
  $\boldsymbol{S} \leftarrow (1-\eta)\,\boldsymbol{S} + \eta\,\boldsymbol{v}(\boldsymbol{v}^*\boldsymbol{\Omega})$ ▷ Update sketch of matrix

function NystromSketch.Reconstruct()
  $\sigma \leftarrow \sqrt{n}\,\texttt{eps}(\texttt{norm}(\boldsymbol{S}))$ ▷ Compute a shift parameter
  $\boldsymbol{S}_\sigma \leftarrow \boldsymbol{S} + \sigma\,\boldsymbol{\Omega}$ ▷ Implicitly form sketch of $\boldsymbol{X} + \sigma\,\mathbf{I}$
  $\boldsymbol{C} \leftarrow \texttt{chol}(\boldsymbol{\Omega}^*\boldsymbol{S}_\sigma)$
  $(\boldsymbol{U}, \boldsymbol{\Sigma}, \sim) \leftarrow \texttt{svd}(\boldsymbol{S}_\sigma/\boldsymbol{C})$ ▷ Dense SVD
  $\boldsymbol{\Lambda} \leftarrow \max\{0, \boldsymbol{\Sigma}^2 - \sigma\,\mathbf{I}\}$ ▷ Remove shift

5.3. A priori error bounds. The Nyström approximation $\widehat{\boldsymbol{X}}$ yields a provably good estimate for the matrix $\boldsymbol{X}$ contained in the sketch [100, Thm. 4.1].

Fact 5.2 (Nyström sketch: Error bound). *Fix a psd matrix $\boldsymbol{X} \in \mathbb{S}_n$. Let $\boldsymbol{S} = \boldsymbol{X}\boldsymbol{\Omega}$ where $\boldsymbol{\Omega} \in \mathbb{F}^{n\times R}$ is standard normal. For each $r < R$, the Nyström approximation (5.3) satisfies*

$$\mathbb{E}_{\boldsymbol{\Omega}}\,\|\boldsymbol{X} - \widehat{\boldsymbol{X}}\|_* \le \left(1 + \frac{r}{R-r-1}\right)\|\boldsymbol{X} - [\![\boldsymbol{X}]\!]_r\|_*, \tag{5.4}$$

where $\mathbb{E}_{\boldsymbol{\Omega}}$ is the expectation with respect to $\boldsymbol{\Omega}$. If we replace $\widehat{\boldsymbol{X}}$ with its rank-r truncation $[\![\widehat{\boldsymbol{X}}]\!]_r$, the error bound (5.4) remains valid. Similar results hold with high probability.

There is a lot more to say about the implementation and behavior of low-rank matrix approximation from streaming data. See Appendix B and our papers [100, 101, 102].

6. Scalable semidefinite programming via SketchyCGAL. We may now present an extension of CGAL that solves the model problem (2.2) with controlled storage and computation.

6.1. The opportunity. Our new algorithm SketchyCGAL augments the CGAL iteration from subsection 3.4. Instead of storing the matrix $\boldsymbol{X}_t$ in the CGAL iteration...

1. We drive the iteration with the d-dimensional primal state variable $\boldsymbol{z}_t := \mathcal{A}\boldsymbol{X}_t$.
2. We maintain a Nyström sketch of the primal iterate $\boldsymbol{X}_t$ using storage $\Theta(Rn)$.
3. When the iteration is halted, say at step T, we use the sketch to construct a rank-R approximation $\widehat{\boldsymbol{X}}_T$ of the implicitly computed solution $\boldsymbol{X}_T$ of the model problem.

As we will see, the resulting method exhibits almost the same convergence behavior as the CGAL algorithm, but it also enjoys a strong storage guarantee.

6.2. The SketchyCGAL iteration. To develop the SketchyCGAL iteration, we begin with the CGAL iteration. Then make the substitutions $\boldsymbol{z}_t = \mathcal{A}\boldsymbol{X}_t$ and $\boldsymbol{h}_t = \mathcal{A}\boldsymbol{H}_t$ to eliminate the matrix variables. Let us summarize what happens; see subsection 6.3 for additional explanation. Algorithm 6.1 contains pseudocode with implementation recommendations.

6.2.1. Initialization. First, we choose the size R of the Nyström sketch. Then draw and fix the random test matrix $\boldsymbol{\Omega} \in \mathbb{F}^{n\times R}$. Select an initial smoothing parameter β_0 and a bound K on the dual variable. Define the smoothing and step size parameters:

$$\beta_t := \beta_0\sqrt{t+1} \quad \text{and} \quad \eta_t := \frac{2}{t+1}. \tag{6.1}$$

Initialize the primal state variable, the sketch, and the dual variable:

$$\boldsymbol{z}_1 := \boldsymbol{0} \in \mathbb{R}^d \quad \text{and} \quad \boldsymbol{S}_1 := \boldsymbol{0} \in \mathbb{F}^{n\times R} \quad \text{and} \quad \boldsymbol{y}_1 := \boldsymbol{0} \in \mathbb{R}^d. \tag{6.2}$$

These choices correspond to the simplest initial iterate $\boldsymbol{X}_1 := \boldsymbol{0}$.

6.2.2. Primal updates. At iteration $t = 1, 2, 3, \dots$, we compute a unit-norm vector $\boldsymbol{v}_t$ that is an approximate minimum eigenvector of the gradient $\boldsymbol{D}_t$ of the smoothed objective:

$$\boldsymbol{v}_t^*\boldsymbol{D}_t\boldsymbol{v}_t \le \lambda_{\min}(\boldsymbol{D}_t) + \frac{\|\boldsymbol{D}_t\|}{\sqrt{t+1}} \quad \text{where} \quad \boldsymbol{D}_t := \boldsymbol{C} + \mathcal{A}^*(\boldsymbol{y}_t + \beta_t(\boldsymbol{z}_t - \boldsymbol{b})). \tag{6.3}$$

This calculation corresponds with (3.9). Form the primal update direction $\boldsymbol{h}_t = \mathcal{A}(\alpha\boldsymbol{v}_t\boldsymbol{v}_t^*)$, and then update the primal state variable $\boldsymbol{z}_t$ and the sketch $\boldsymbol{S}_t$:

$$\boldsymbol{z}_{t+1} := (1-\eta_t)\boldsymbol{z}_t + \eta_t\boldsymbol{h}_t \quad \text{and} \quad \boldsymbol{S}_{t+1} := (1-\eta_t)\boldsymbol{S}_t + \eta_t\alpha\boldsymbol{v}_t(\boldsymbol{v}_t^*\boldsymbol{\Omega}). \tag{6.4}$$

We obtain the update rule for the primal state variable $\boldsymbol{z}_t$ by applying the linear map $\mathcal{A}$ to the primal update rule (3.6). The update rule for the sketch $\boldsymbol{S}_t$ follows from (5.2).

6.2.3. Dual updates. The update to the dual variable takes the form

$$\boldsymbol{y}_{t+1} = \boldsymbol{y}_t + \gamma_t(\boldsymbol{z}_{t+1} - \boldsymbol{b}) \tag{6.5}$$

where we choose the largest γ_t that satisfies the conditions

$$\gamma_t\|\boldsymbol{z}_{t+1} - \boldsymbol{b}\|^2 \le \frac{4\alpha^2\beta_0}{(t+1)^{3/2}}\|\mathcal{A}\|^2 \quad \text{and} \quad 0 \le \gamma_t \le \beta_0. \tag{6.6}$$

If needed, we set $\gamma_t = 0$ to prevent $\|\boldsymbol{y}_{t+1}\| > K$. This is the SketchyCGAL iteration.

6.3. Connection with CGAL. There is a tight connection between the iterates of SketchyCGAL and CGAL. Let $\boldsymbol{X}_1 := \boldsymbol{0}$. Using the vectors $\boldsymbol{v}_t$ computed in (6.3), define matrices

$$\boldsymbol{H}_t := \alpha\boldsymbol{v}_t\boldsymbol{v}_t^* \quad \text{and} \quad \boldsymbol{X}_{t+1} := (1-\eta_t)\boldsymbol{X}_t + \eta_t\boldsymbol{H}_t. \tag{6.7}$$

With these definitions, the following loop invariants are in force:

$$\boldsymbol{h}_t = \mathcal{A}\boldsymbol{H}_t \quad \text{and} \quad \boldsymbol{z}_t = \mathcal{A}\boldsymbol{X}_t \quad \text{and} \quad \boldsymbol{S}_t = \boldsymbol{X}_t\boldsymbol{\Omega}. \tag{6.8}$$

By comparing subsections 3.4 and 6.2, we see that the trajectory $\{(\boldsymbol{X}_t, \boldsymbol{H}_t, \boldsymbol{y}_t) : t = 1, 2, 3, \dots\}$ could also have been generated by running the CGAL iteration. In other words, the variables in SketchyCGAL track the variables of some invocation of CGAL and inherit their behavior. We refer to the matrices $\boldsymbol{X}_t$ as the *implicit* CGAL iterates.

Algorithm 6.1 SketchyCGAL for the model problem (2.2)

Input: Problem data for (2.2) implemented via the primitives (2.4), sketch size R, number T of iterations
Output: Rank-R approximate solution to (2.2) in factored form $\widehat{\boldsymbol{X}}_T = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^*$ where $\boldsymbol{U} \in \mathbb{F}^{n\times R}$ has orthonormal columns and $\boldsymbol{\Lambda} \in \mathbb{R}^{R\times R}$ is nonnegative diagonal
Recommendation: To achieve (2.3), set R as large as possible, and set $T \approx \varepsilon^{-1}$

function SketchyCGAL(R; T)
  Scale problem data (subsection 7.1.2) ▷ **[opt]** Recommended!
  $\beta_0 \leftarrow 1$ and $K \leftarrow +\infty$ ▷ Default parameters
  NystromSketch.Init(n, R)
  $\boldsymbol{z} \leftarrow \mathbf{0}_d$ and $\boldsymbol{y} \leftarrow \mathbf{0}_d$
  for $t \leftarrow 1, 2, 3, \ldots, T$ **do**
    $\beta \leftarrow \beta_0\sqrt{t+1}$ and $\eta \leftarrow 2/(t+1)$
    $(\xi, \boldsymbol{v}) \leftarrow$ ApproxMinEvec($\boldsymbol{C} + \mathcal{A}^*(\boldsymbol{y} + \beta(\boldsymbol{z} - \boldsymbol{b}))$; q_t) ▷ Use primitives (2.4)❶❷
      ▷ With Algorithm 4.1, set $q_t = 8t^{1/2}\log n$.
      ▷ With Algorithm 4.2, set $q_t = t^{1/4}\log n$
    $\boldsymbol{z} \leftarrow (1-\eta)\,\boldsymbol{z} + \eta\,\mathcal{A}(\alpha\boldsymbol{v}\boldsymbol{v}^*)$ ▷ Use primitive (2.4)❸
    $\boldsymbol{y} \leftarrow \boldsymbol{y} + \gamma(\boldsymbol{z} - \boldsymbol{b})$ ▷ γ is the largest solution to (6.6)
    NystromSketch.RankOneUpdate($\sqrt{\alpha}\boldsymbol{v}$, η)
  $(\boldsymbol{U}, \boldsymbol{\Lambda}) \leftarrow$ NystromSketch.Reconstruct
  $\boldsymbol{\Lambda} \leftarrow \boldsymbol{\Lambda} + (\alpha - \operatorname{tr}(\boldsymbol{\Lambda}))\mathbf{I}_R/R$ ▷ **[opt]** Enforce trace constraint in (2.2)

6.3.1. Approximating the CGAL iterates. We do not have access to the implicit CGAL iterates described above. Nevertheless, the sketch permits us to approximate them! After iteration t of SketchyCGAL, we can form a rank-R approximation $\widehat{\boldsymbol{X}}_t$ of the implicit iterate $\boldsymbol{X}_t$ by invoking the formula (5.3) with $\boldsymbol{S} = \boldsymbol{S}_t$. According to Fact 5.2, for each $r < R$,

$$\mathbb{E}_{\boldsymbol{\Omega}}\,\|\boldsymbol{X}_t - \widehat{\boldsymbol{X}}_t\|_* \le \left(1 + \frac{r}{R - r - 1}\right)\|\boldsymbol{X}_t - [\![\boldsymbol{X}_t]\!]_r\|_*. \tag{6.9}$$

In other words, the computed approximation $\widehat{\boldsymbol{X}}_t$ is a good proxy for the implicit iterate $\boldsymbol{X}_t$ whenever the latter matrix is well-approximated by a low-rank matrix. The same bound (6.9) holds if we replace $\widehat{\boldsymbol{X}}_t$ by the truncated rank-r matrix $[\![\widehat{\boldsymbol{X}}_t]\!]_r$.

Remark 6.1 (Trace constraint). The model problem (2.2) requires the trace of the matrix variable to equal α, but the computed solution $\widehat{\boldsymbol{X}}_t$ rarely satisfies this constraint. Algorithm 6.1 includes an optional step that corrects the trace of the computed solution. Although this step is principled and justifiable, our theoretical analysis *does not* include its effects.

6.3.2. Solution quality. Given the quantities computed by SketchyCGAL, we can assess how well the implicit iterate $\boldsymbol{X}_t$ solves the model problem (2.2). See Appendix C.1.

6.4. Convergence of SketchyCGAL. The implicit iterates $\boldsymbol{X}_T$ converge to a solution of the model problem (2.2) at the same rate as the iterates of CGAL would. On average, the

rank-R iterates $\widehat{\boldsymbol{X}}_T$ track the implicit iterates. The discrepancy between them depends on how well the implicit iterates are approximated by low-rank matrices. Here is a simple convergence result that reflects this intuition; see Appendix C.2 for the proof and further results.

Theorem 6.2 (SketchyCGAL: Convergence). *Assume problem (2.2) satisfies strong duality, and let $\Psi_\star$ be its solution set. For each $r < R$, the iterates $\widehat{\boldsymbol{X}}_t$ computed by SketchyCGAL (subsections 6.2 and 6.3) satisfy*

$$\limsup_{t\to\infty} \mathbb{E}_{\boldsymbol{\Omega}} \operatorname{dist}_*(\widehat{\boldsymbol{X}}_t, \Psi_\star) \leq \left(1 + \frac{r}{R-r-1}\right) \cdot \max_{\boldsymbol{X}\in\Psi_\star} \|\boldsymbol{X} - [\![\boldsymbol{X}]\!]_r\|_*.$$

The same bound holds if we replace $\widehat{\boldsymbol{X}}_t$ with the truncated approximation $[\![\widehat{\boldsymbol{X}}_t]\!]_r$. Here, dist_* is the nuclear-norm distance between a matrix and a set of matrices.

6.5. Resource usage. We may now account for the resources required by SketchyCGAL.

6.5.1. Working storage. The main working variables in the SketchyCGAL iteration are the primal state $\boldsymbol{z}_t \in \mathbb{R}^d$, the computed eigenvector $\boldsymbol{v}_t \in \mathbb{R}^n$, the update direction $\boldsymbol{h}_t \in \mathbb{R}^d$, the test matrix $\boldsymbol{\Omega} \in \mathbb{R}^{n\times R}$, the sketch $\boldsymbol{S}_t \in \mathbb{R}^{n\times R}$, and the dual variable $\boldsymbol{y}_t \in \mathbb{R}^d$. The total cost for storing these variables is $\Theta(d + Rn)$.

6.5.2. Approximate eigenvectors. The bottleneck in SketchyCGAL is the approximate eigenvector computation (6.3). For the subproblem in iteration t, we have two options:

1. **Shifted power.** Perform $q_t = O(t^{1/2}\log(tn))$ steps of Algorithm 4.1, expending $O(q_t)$ applications of primitives (2.4)❶❷, plus $O(q_t n)$ arithmetic and $O(n)$ storage.
2. **Randomized Lanczos.** Do $q_t = O(t^{1/4}\log(tn))$ steps of Algorithm 4.2, using $O(q_t)$ applications of primitives (2.4)❶❷, plus $O(q_t n + q_t^2)$ arithmetic and $O(q_t n)$ storage.

These choices ensure that, with constant probability, the eigenvector computation succeeds in every iteration t of SketchyCGAL. In a practical implementation, we undertake somewhat fewer steps in the eigenvector algorithms; see subsections 4.2 and 4.3.

6.5.3. Variable updates. The remaining computation takes place in the variable updates. To form the primal update direction $\boldsymbol{h}_t$, we invoke the primitive (2.4)❸. To update the primal state variable $\boldsymbol{z}_t$ and the sketch $\boldsymbol{S}_t$ in (6.4), we need $O(d + Rn)$ arithmetic operations. No further storage is required at this stage.

6.5.4. Overview of costs. Table 1 documents the cost of performing T iterations of the SketchyCGAL method. The first column summarizes the resources consumed in the outer iteration. The second column lists the additional resource usage if we invoke the shifted power method (Algorithm 4.1) to solve the approximate eigenvalue problem (6.3). The third column lists the extra resources if instead we invoke the Lanczos method (Algorithm 4.2).

In light of Fact 3.1 and subsection 6.3, we can be confident that the implicit iterate $\boldsymbol{X}_T$ is ε-optimal within $T = O(\varepsilon^{-2})$ iterations. The formula (6.9) gives *a priori* guarantees on the quality of the approximation $\widehat{\boldsymbol{X}}_T$.

6.6. Theoretical performance of SketchyCGAL. We can package up this discussion in a theorem that describes the theoretical performance of the SketchyCGAL method.

Table 1

*Resource usage for T iterations of SketchyCGAL with sketch size R using two different eigensolvers. The algorithm returns a rank-R approximation to an ε-optimal solution (subsection 2.2) to the model problem (2.2). **In theory,** $T \sim \varepsilon^{-2}$. **In practice,** $T \sim \varepsilon^{-1}$. Constants and logarithms are omitted. See subsection 6.5.4.*

	SketchyCGAL Algorithm 6.1	w/shifted power Algorithm 4.1	**or** w/Lanczos Algorithm 4.2
Storage (floats)	$d + Rn$	n	$T^{1/4}n$
Arithmetic (flops)	$T(d + Rn)$	$T^{3/2}n$	$T^{5/4}n + T^{3/2}$
Primitives (calls)			
(2.4)❶❷	—	$T^{3/2}$	$T^{5/4}$
(2.4)❸	T	—	—

Theorem 6.3 (SketchyCGAL). *Assume the model problem (2.2) satisfies strong duality. Fix a rank r, and set the sketch size $R \geq \zeta^{-1}r + 1$. After $T = O(\varepsilon^{-2})$ iterations, SketchyCGAL (subsections 6.2 and 6.3) returns a rank-r approximation $\widehat{\boldsymbol{X}} = [\![\widehat{\boldsymbol{X}}_T]\!]_r$ to an ε-optimal point of (2.2) that satisfies (2.3) with constant probability.*

1. ***Shifted power.*** *Suppose we use Algorithm 4.1 for the subproblem (6.3). The storage cost is $O(d + Rn)$ floats. Arithmetic costs are $O(\varepsilon^{-2}(d + Rn) + \varepsilon^{-3}n\log(n/\varepsilon))$ flops, $O(\varepsilon^{-3}\log(n/\varepsilon))$ invocations of primitives (2.4)❶❷, and $O(\varepsilon^{-2})$ applications of (2.4)❸.*
2. ***Lanczos.*** *Suppose we use Algorithm 4.2 to solve (6.3). Then the storage cost is $O(d + Rn + \varepsilon^{-1/2}n\log(n/\varepsilon))$ floats. The arithmetic requirements are $O(\varepsilon^{-2}(d + Rn) + (\varepsilon^{-3/2}n + \varepsilon^{-3})\log(n/\varepsilon))$ flops, $O(\varepsilon^{-5/2}\log(n/\varepsilon))$ invocations of the primitives (2.4)❶❷, and $O(\varepsilon^{-2})$ applications of the primitive (2.4)❸.*

In practice, SketchyCGAL performs substantially better than Theorem 6.3 suggests: empirically, we find that $T = O(\varepsilon^{-1})$. See section 7 for numerical evidence.

6.7. Comparison with CGAL. Like CGAL, SketchyCGAL is a general SDP solver with strong guarantees. Yet our changes lead to a new method with a better computational profile.

We motivated sketching as a mechanism to control storage. This improvement is visible when we compare the $\Theta(n^2)$ storage cost of CGAL to the $\Theta(d + Rn)$ storage cost of SketchyCGAL. But sketching also reduces arithmetic and communication. Indeed, SketchyCGAL works with state vectors of length d, and it updates the $\Theta(Rn)$ entries of the sketch $\boldsymbol{S}_t$ at each iteration. Meanwhile, CGAL must update all $\Theta(n^2)$ entries of the matrix iterate $\boldsymbol{X}_t$.

As a consequence of these improvements, SketchyCGAL scales to problems that are orders of magnitude larger than CGAL can handle. See section 7.

7. Numerical examples. This section showcases computational experiments that establish that SketchyCGAL is a practical method for solving large SDPs. We show that the algorithm is **flexible** by applying it to several classes of SDPs, and we show it is **reliable** by solving a large number of instances of each type. We give empirical evidence that the (implicit) iterates are **converging** to optimality much faster than Theorem 6.3 suggests. Explicit comparisons with other general SDP solvers demonstrate that SketchyCGAL is **competitive** for small SDPs, while it **scales** to problems that standard methods cannot handle.

7.1. Setup. We begin with an overview of the experimental regimen.

7.1.1. Computational environment. All experiments are performed in MATLAB_R2018a with double-precision arithmetic (8 bytes per real float). Source code is included with the supplementary material. To simulate the processing power of a personal computer, we use a single Intel Xeon CPU E5-2630 v3, clocked at 2.40 GHz, with RAM usage capped at 16 GB.

Arithmetic costs are measured in terms of actual CPU time. MATLAB does not currently offer a memory profiler, so we externally monitor the total memory allocated. We approximate the storage cost by reporting the peak value minus the storage at MATLAB's idle state.

7.1.2. Problem scaling. The bounds in the convergence theorem Fact 3.1 for the implicit iterates of SketchyCGAL depend on problem scaling. Our analysis motivates us to set

$$\|\boldsymbol{C}\|_{\mathrm{F}} = \|\mathcal{A}\| = \alpha = 1 \quad \text{and} \quad \|\boldsymbol{A}_1\|_{\mathrm{F}} = \|\boldsymbol{A}_2\|_{\mathrm{F}} = \cdots = \|\boldsymbol{A}_d\|_{\mathrm{F}}. \tag{7.1}$$

In our experiments, we enforce the scalings (7.1), except where noted.

7.1.3. Quality of solutions. Given a prospective solution $\boldsymbol{X}$ of (2.2), we compute the relative suboptimality and feasibility as

$$\texttt{objective residual} = \frac{|\langle \boldsymbol{C}, \boldsymbol{X}\rangle - \langle \boldsymbol{C}, \boldsymbol{X}_\star\rangle|}{\max\{1, |\langle \boldsymbol{C}, \boldsymbol{X}_\star\rangle|\}} \quad \text{and} \quad \texttt{infeasibility} = \frac{\|\mathcal{A}\boldsymbol{X} - \boldsymbol{b}\|}{\max\{1, \|\boldsymbol{b}\|\}}.$$

It is standard to place the maximum in each denominator to handle small values gracefully. These quantities are evaluated with respect to the original (not rescaled) problem data.

7.1.4. Implementation. Our experiments require a variant of SketchyCGAL that can handle inequality constraints; see Appendix D. We implement the algorithm with the default parameters ($\beta_0 = 1$ and $K = +\infty$), and the sketch uses a Gaussian test matrix. The eigenvalue subproblem is solved via randomized Lanczos (Algorithm 4.2). We include the optional trace normalization (line 13 in Algorithm 6.1) whenever appropriate. No further tuning is done.

7.2. The MaxCut SDP. We begin with MaxCut SDPs (1.3). Our goal is to assess the storage and arithmetic costs of SketchyCGAL for a standard testbed. We compare with provable solvers for general SDPs: SEDUMI [97], SDPT3 [99], MOSEK [78], and SDPNAL+ [108].

7.2.1. Rounding. To extract a cut from an approximate solution to (1.3), we apply a simple rounding procedure. SketchyCGAL returns a matrix $\widehat{\boldsymbol{X}} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^*$, where $\boldsymbol{U} \in \mathbb{R}^{n\times R}$ has orthonormal columns. The columns of the entrywise signum, $\operatorname{sgn}(\boldsymbol{U})$, are the signed indicators of R cuts. We compute the weights of all R cuts and select the largest. The other solvers return a full-dimensional solution $\widehat{\boldsymbol{X}}$; we compute the top R eigenvectors of $\widehat{\boldsymbol{X}}$ using the MATLAB command `eigs` and invoke the same rounding procedure.

7.2.2. Datasets. We consider datasets from two different benchmark groups:

1. GSET: 67 binary-valued matrices generated by an autonomous random graph generator and published online [109]. The dimension n varies from 800 to 10 000.
2. DIMACS10: This benchmark consists of 152 symmetric matrices (with n varying from 39 to 50 912 018) chosen for the 10th DIMACS Implementation Challenge [13]. We consider the 138 datasets with dimension $n \leq 8\,000\,000$.

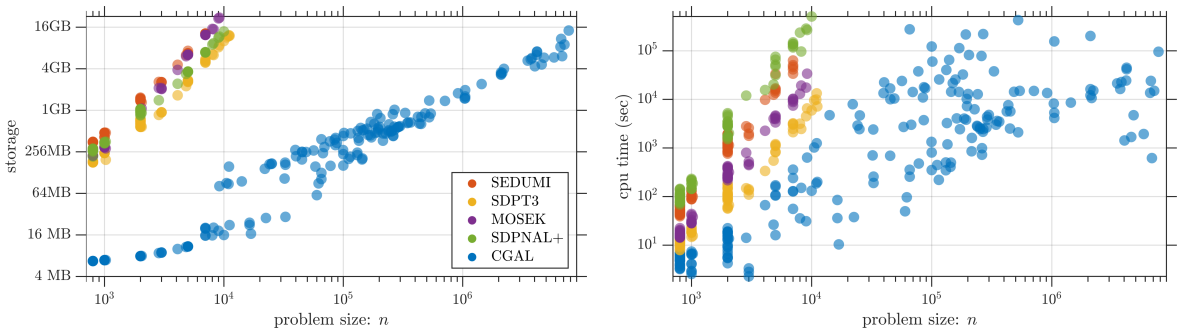


Figure 7.1. MaxCut SDP: Scalability. *Storage cost [left] and runtime [right] of SketchyCGAL with sketch size $R = 10$ as compared with four standard SDP solvers. Each marker is one dataset. See subsection 7.2.3.*

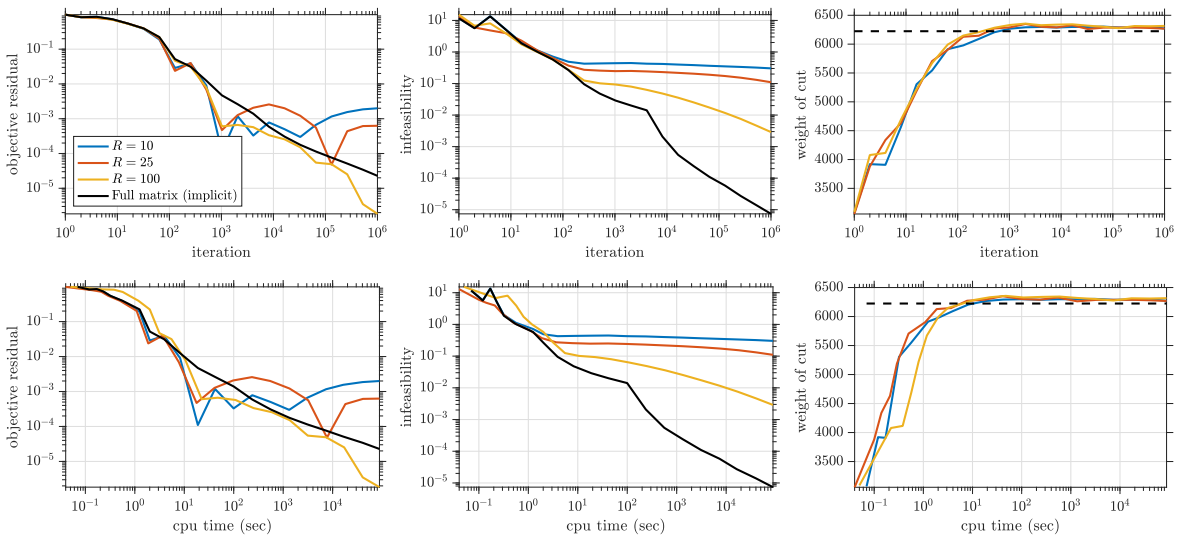


Figure 7.2. MaxCut SDP: Convergence. *We solve the MaxCut SDP for the G67 dataset ($n = 10\,000$) with SketchyCGAL. The subplots show the suboptimality [left], infeasibility [center], and cut value [right] of the implicit iterates and low-rank approximations for sketch size $R \in \{10, 25, 100\}$ as a function of iteration [top] and CPU time [bottom]. The dashed line is the cut value of a high-accuracy SDPT3 solution. See subsection 7.2.4.*

7.2.3. Storage and arithmetic comparisons. We run each solver for each dataset and measure the storage cost and the runtime. We invoke SketchyCGAL with rank parameter $R = 10$, and we stop the algorithm when the error bound described in Appendix C.1 guarantees that the implicit iterates have both suboptimality and feasibility below 10^{-4}. We use other solvers with their default parameter settings and stopping criteria.

Figure 7.1 displays the results of this experiment. The conventional convex solvers do not scale beyond $n = 10^4$ due to their high storage costs. In contrast, SketchyCGAL handles problems with $n \approx 8 \cdot 10^6$. Further results appear in Appendix F.1.

7.2.4. Empirical convergence rates. Next, we investigate the empirical convergence of SketchyCGAL and the effect of the sketch size parameter R. We consider the MaxCut SDP (1.3) for the G67 dataset ($n = 10\,000$), the largest instance in GSET, and we use a high-accuracy

solution from SDPT3 to approximate the optimal point. We run 10^6 iterations of SketchyCGAL for each $R \in \{10, 25, 100\}$.

Figure 7.2 illustrates the convergence of SketchyCGAL for this problem. After t iterations, the residual and infeasibility decline as $O(t^{-1})$, which is far better than the $O(t^{-1/2})$ bound in Theorem 6.3. Similar behavior is manifest for other datasets and other problems.

Figure 7.2 also displays the weight of the cut obtained after rounding, compared with the weight of the cut obtained from the SDPT3 solution. Observe that sketch size $R = 10$ is sufficient, and the SketchyCGAL solutions yield excellent cuts after a few hundred iterations.

7.2.5. Hard MaxCut instances. Waldspurger & Waters [105] construct instances of the MaxCut SDP (1.3) that are challenging for algorithms based on the Burer–Monteiro [27] factorization method (subsection 8.6). Each instance has a unique solution and the solution has rank 1, but Burer–Monteiro methods require factorization rank $R = \Theta(\sqrt{n})$, resulting in storage cost $\Theta(n^{3/2})$. In Appendix F.2, we give numerical evidence that SketchyCGAL solves these instances with sketch size $R = 2$, achieving the optimal storage $\Theta(n)$.

7.3. Abstract phase retrieval. Phase retrieval is the problem of reconstructing a complex-valued signal from intensity-only measurements. It arises in interferometry [41], speech processing [16], array imaging [33], microscopy [55], and many other applications. We will outline a standard method [33, 55] for performing phase retrieval by means of an SDP.

This section uses synthetic instances of a phase retrieval SDP to compare the scaling behavior of SketchyCGAL and CGAL. We also consider a third algorithm ThinCGAL, inspired by [112], that maintains a thin SVD of the matrix variable via rank-one updates [25].

7.3.1. Phase retrieval SDPs. Let $\boldsymbol{\chi}_\natural \in \mathbb{C}^n$ be an unknown (discrete) signal. For known vectors $\boldsymbol{a}_i \in \mathbb{C}^n$, we acquire measurements of the form

$$b_i = |\langle \boldsymbol{a}_i, \boldsymbol{\chi}_\natural \rangle|^2 \quad \text{for} \quad i = 1, 2, 3, \dots, d. \tag{7.2}$$

Abstract phase retrieval is the challenging problem of recovering $\boldsymbol{\chi}_\natural$ from $\boldsymbol{b}$.

Let us summarize a lifting approach introduced by Balan et al. [15]. The key idea is to replace the signal vector $\boldsymbol{x}_\natural$ by the matrix $\boldsymbol{X}_\natural = \boldsymbol{\chi}_\natural \boldsymbol{\chi}_\natural^*$. Then rewrite (7.2) as

$$b_i = \boldsymbol{a}_i^* \boldsymbol{X}_\natural \boldsymbol{a}_i = \langle \boldsymbol{A}_i, \boldsymbol{X}_\natural \rangle \quad \text{where} \quad \boldsymbol{A}_i = \boldsymbol{a}_i \boldsymbol{a}_i^* \quad \text{for } i = 1, 2, 3, \dots, d. \tag{7.3}$$

Promoting the implicit constraints on $\boldsymbol{X}_\natural$ and forming the $\boldsymbol{A}_i$ into a linear map $\mathcal{A}$, we can express the problem of finding $\boldsymbol{X}_\natural$ as a feasibility problem with a matrix variable:

$$\text{find} \quad \boldsymbol{X} \in \mathbb{S}_n \quad \text{subject to} \quad \mathcal{A}\boldsymbol{X} = \boldsymbol{b}, \quad \boldsymbol{X} \text{ is psd}, \quad \operatorname{rank}(\boldsymbol{X}) = 1.$$

To reach a tractable convex formulation, we pass to a trace minimization SDP [40]:

$$\text{minimize} \quad \operatorname{tr} \boldsymbol{X} \quad \text{subject to} \quad \mathcal{A}\boldsymbol{X} = \boldsymbol{b}, \quad \boldsymbol{X} \text{ is psd}, \quad \operatorname{tr} \boldsymbol{X} \le \alpha. \tag{7.4}$$

The parameter α is an upper bound on the signal energy $\|\boldsymbol{\chi}_\natural\|^2$, which can be estimated from the observed data $\boldsymbol{b}$; see [112] for the details. We can solve the SDP (7.4) via SketchyCGAL.

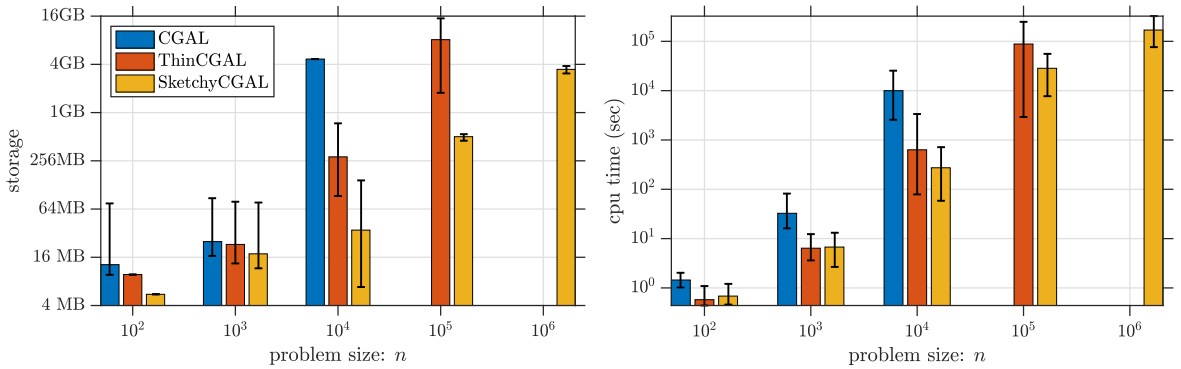


Figure 7.3. Phase retrieval SDP: Scalability. *Storage cost [left] and runtime [right] to solve random instances with algorithms CGAL, ThinCGAL, and SketchyCGAL. Height of the bar is the mean and the interval marks are the min/max over 20 trials. Missing bars indicate the failure. See subsection 7.3.4.*

7.3.2. Rounding. Suppose that we have obtained an approximate solution $\boldsymbol{X}$ to (7.4). To estimate the signal $\boldsymbol{\chi}_\natural$, we form the vector $\boldsymbol{\chi} = \sqrt{\lambda}\boldsymbol{u}$ where $(\lambda, \boldsymbol{u})$ is a maximum eigenpair of $\boldsymbol{X}$. Both SketchyCGAL and ThinCGAL return eigenvalue decompositions, so this step is trivial. For CGAL, we use the MATLAB function `eigs` to perform this computation.

7.3.3. Dataset and evaluation. We consider synthetic phase retrieval instances. For each $n \in \{10^2, 10^3, \dots, 10^6\}$, we generate 20 independent datasets as follows. First, draw $\boldsymbol{\chi}_\natural \in \mathbb{C}^n$ from the complex standard normal distribution. Then acquire $d = 10n$ phaseless measurements (7.2) using the coded diffraction pattern model [31]; see Appendix E.1. The induced linear maps $\mathcal{A}$ and $\mathcal{A}^*$ can be applied via the fast Fourier transform (FFT).

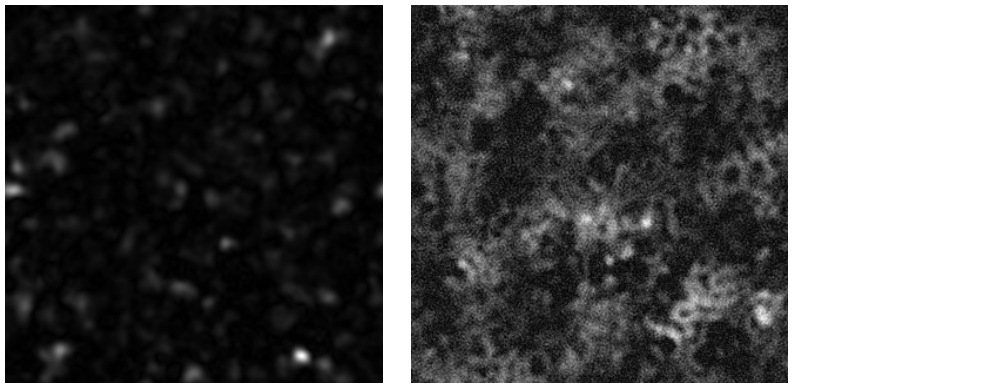
The relative error in a signal reconstruction $\boldsymbol{\chi}$ is given by $\min_{\phi\in\mathbb{R}} \|\mathrm{e}^{i\phi}\boldsymbol{\chi} - \boldsymbol{\chi}_\natural\|/\|\boldsymbol{\chi}_\natural\|$. In the SDP (7.4), we set $\alpha = 3n$ to demonstrate insensitivity of the algorithm to the parameter.

7.3.4. Storage and arithmetic comparisons. For each algorithm, we report the storage cost and runtime required to produce a signal estimate $\boldsymbol{\chi}$ with (exact) relative error below 10^{-2}. We invoke SketchyCGAL with sketch size parameter $R = 5$.

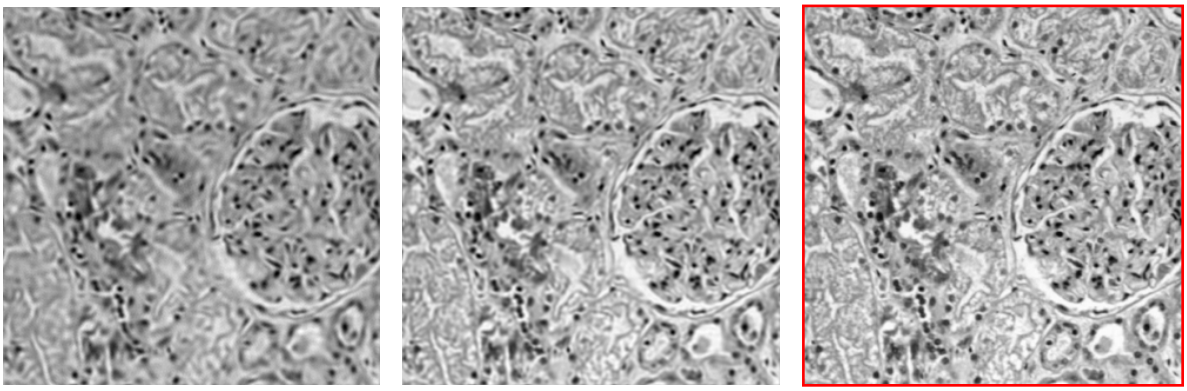
Figure 7.3 displays the outcome. We witness the benefit of sketching for both storage and arithmetic. CGAL fails for all large-scale instances ($n = 10^5$ and 10^6) due to storage allocation. For the same reason, ThinCGAL solves 19 of 20 instances for $n = 10^5$; it always fails for $n = 10^6$. SketchyCGAL successfully solves all problem instances to the target accuracy.

7.4. Phase retrieval in microscopy. Next, we study a more realistic phase retrieval problem that arises from a type of microscopy system [119] called Fourier ptychography (FP). Phase retrieval SDPs offer a potential approach to FP imaging [55]. This section shows that SketchyCGAL can successfully solve the difficult phase retrieval SDPs that arise from FP.

7.4.1. Fourier ptychography. FP microscopes circumvent the physical limits of a simple lens to achieve high-resolution and wide field-of-view simultaneously [119]. To do so, an FP microscope illuminates a sample from many angles and uses a simple lens to collect low-resolution intensity-only images. The measurements are low-pass filters, whose transfer functions depend on the lens and the angle of illumination [55]. From the data, we form a



(a) $t = 10$ (30 sec) (b) $t = 100$ (415 sec)



(c) $t = 1\,000$ (6 714 sec) (d) $t = 10\,000$ (101 358 sec) (e) original

Figure 7.4. Phase retrieval SDP: Imaging. *Reconstruction of an $n = 320^2$ pixel image from Fourier ptychography data. We solve an $n \times n$ phase retrieval SDP via SketchyCGAL with rank parameter $R = 5$ and show the images obtained at iterations $t = 10, 10^2, 10^3, 10^4$. The last subfigure is the original. See subsection 7.4.3.*

high-resolution image by solving a phase retrieval problem; e.g., via the SDP (7.4).

The high-resolution image of the sample is represented by a Fourier-domain vector $\boldsymbol{\chi}_\natural \in \mathbb{C}^n$. We acquire d intensity-only measurements of the form (7.2), where d is the total number of pixels in the low-resolution illuminations. The low-pass measurements are encoded in vectors $\boldsymbol{a}_i$. The operators $\mathcal{A}$ and $\mathcal{A}^*$, built from the matrices $\boldsymbol{A}_i = \boldsymbol{a}_i \boldsymbol{a}_i^*$, can be applied via the FFT.

7.4.2. Dataset and evaluation. The authors of [55] provided transmission matrices $\boldsymbol{A}_i$ of a working FP system. We simulate this system in the computer environment to acquire noiseless intensity-only measurements of a high-resolution target image [36]. In this setup, $\boldsymbol{\chi}_\natural \in \mathbb{C}^n$ corresponds to the Fourier transform of an $n = 320^2 = 102\,400$ pixel grayscale image. We normalize $\boldsymbol{\chi}_\natural$ so that $\|\boldsymbol{\chi}_\natural\| = 1$. We acquire 225 low-resolution illuminations of the original image, each with $64^2 = 4\,096$ pixels. The total number of measurements is $d = 921\,600$.

We evaluate the error of an estimate $\boldsymbol{\chi}$ as in subsection 7.3.3. Although we know that the signal energy equals 1, it is more realistic to approximate the signal energy by setting $\alpha = 1.5$ in the SDP (7.4).

7.4.3. FP imaging. We solve the phase retrieval SDP (7.4) by performing 10 000 iterations of SketchyCGAL with rank parameter $R = 5$. The top eigenvector of the output gives an approximation $\boldsymbol{\chi} \in \mathbb{C}^n$ of the signal. The inverse Fourier transform of $\boldsymbol{\chi}$ is the desired image.

Figure 7.4 displays the image reconstruction after $t \in \{10, 10^2, 10^3, 10^4\}$ iterations. We obtain a good-quality result in 2 hours after 1 000 iterations; a sharper image emerges in 28 hours after 10 000 iterations are complete. We believe the computational time can be reduced substantially with a parallel or GPU implementation. Nevertheless, it is gratifying that we have solved an SDP whose matrix variable has over 10^{10} entries! See Appendix E.2 for a larger FP instance with $n = 640^2$ pixels.

7.5. The quadratic assignment problem. The quadratic assignment problem (QAP) is a very difficult combinatorial optimization problem that includes the traveling salesman, max-clique, bandwidth problems, and many others as special cases [74]. SDP relaxations offer a powerful approach for obtaining good solutions to large QAP problems [117]. In this section, we demonstrate that SketchyCGAL can solve these challenging SDPs.

7.5.1. QAP. We begin with the simplest form of the QAP. Fix symmetric $\mathtt{n} \times \mathtt{n}$ matrices $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{S}_{\mathtt{n}}$ where $\mathtt{n}$ is a natural number. We wish to "align" the matrices by solving

$$\text{minimize} \quad \operatorname{tr}(\boldsymbol{A}\boldsymbol{\Pi}\boldsymbol{B}\boldsymbol{\Pi}^*) \quad \text{subject to} \quad \boldsymbol{\Pi} \text{ is an } \mathtt{n} \times \mathtt{n} \text{ permutation matrix.} \tag{7.5}$$

Recall that a permutation matrix $\boldsymbol{\Pi}$ has precisely one nonzero entry in each row and column, and that nonzero entry equals one. A brute force search over the $\mathtt{n}!$ permutation matrices of size $\mathtt{n}$ quickly becomes intractable as $\mathtt{n}$ grows. Unsurprisingly, the QAP problem (7.5) is NP-hard [94]. Instances with $\mathtt{n} > 30$ usually cannot be solved in reasonable time.

7.5.2. Relaxations. There is an extensive literature on SDP relaxations for QAP, beginning with the work [117] of Zhao et al. We consider a weaker relaxation inspired by [56, 26]:

$$\begin{aligned} \text{minimize} \quad & \operatorname{tr}[(\boldsymbol{B} \otimes \boldsymbol{A})\mathsf{Y}] \\ \text{subject to} \quad & \operatorname{tr}_1(\mathsf{Y}) = \mathbf{I}, \quad \operatorname{tr}_2(\mathsf{Y}) = \mathbf{I}, \quad \mathcal{G}(\mathsf{Y}) \geq 0, \\ & \operatorname{vec}(\boldsymbol{P}) = \operatorname{diag}(\mathsf{Y}), \quad \boldsymbol{P}\mathbf{1} = \mathbf{1}, \quad \mathbf{1}^*\boldsymbol{P} = \mathbf{1}^*, \quad \boldsymbol{P} \geq \mathbf{0}, \\ & \begin{bmatrix} 1 & \operatorname{vec}(\boldsymbol{P})^* \\ \operatorname{vec}(\boldsymbol{P}) & \mathsf{Y} \end{bmatrix} \succcurlyeq \mathbf{0}, \quad \operatorname{tr}\mathsf{Y} = n. \end{aligned} \tag{7.6}$$

We have written $\otimes$ for the Kronecker product.

The constraint $\mathcal{G}(\mathsf{Y}) \geq 0$ enforces nonnegativity of a subset of the entries in Y. In the Zhao et al. relaxation, $\mathcal{G}$ is the identity map, so it yields $O(\mathtt{n}^4)$ constraints. We reduce the complexity by choosing $\mathcal{G}$ more carefully. In our formulation, $\mathcal{G}$ extracts precisely the nonzero entries of $\boldsymbol{B} \otimes \mathbf{1}\mathbf{1}^*$. This is beneficial because $\boldsymbol{B}$ is sparse in many applications. For example, in traveling salesman and bandwidth problems, $\boldsymbol{B}$ has $O(\mathtt{n})$ nonzero entries, so the map $\mathcal{G}$ produces only $O(\mathtt{n}^3)$ constraints.

The main variable Y in (7.6) has dimension $\mathtt{n}^2 \times \mathtt{n}^2$. As a consequence, the problem has $O(\mathtt{n}^4)$ degrees of freedom, together with $O(\mathtt{n}^3)$ to $O(\mathtt{n}^4)$ constraints (depending on $\mathcal{G}$). The explosive growth of this relaxation scuttles most algorithms by the time $\mathtt{n} > 50$. To solve larger instances, many researchers resort to even weaker relaxations.

In contrast, we can solve the relaxation (7.6) directly using SketchyCGAL, up to $\mathtt{n} = 150$. By limiting the number of inequality constraints, via the operator $\mathcal{G}$, we achieve substantial reductions in resource usage. We validate our algorithm on QAPs where the exact solution is known, and we compare the performance with algorithms [116, 26] for other relaxations.

7.5.3. Rounding. Given an approximate solution to (7.6), we use a rounding method to construct a permutation. SketchyCGAL returns a matrix $\widehat{\boldsymbol{X}} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^*$ where $\boldsymbol{U}$ has dimension $(\mathtt{n}^2 + 1) \times R$. We extract the first column of $\boldsymbol{U}$, discard its first entry and reshape the remaining part into an $\mathtt{n} \times \mathtt{n}$ matrix. Then we project this matrix onto the set of permutation matrices via the Hungarian method [66, 79, 59]. This yields a feasible point $\boldsymbol{\Pi}$ for the problem (7.5). We repeat this procedure for all R columns of $\boldsymbol{U}$, and we pick the one that minimizes $\operatorname{tr}(\boldsymbol{A}\boldsymbol{\Pi}\boldsymbol{B}\boldsymbol{\Pi}^*)$. This permutation gives an upper bound on the optimal value of (7.5).

7.5.4. Datasets and evaluation. We consider instances from QAPLIB [30] and TSPLIB [91] that are used in [26]. The optimal values are known, and the permutation size $\mathtt{n}$ varies between 12 and 150. (Recall that the SDP matrix dimension $n = \mathtt{n}^2 + 1$.) We report

$$\texttt{relative gap } \% = \frac{\texttt{upper bound obtained - optimum}}{\texttt{optimum}} \times 100 \tag{7.7}$$

7.5.5. Solving QAPs. To solve (7.6), we cannot use the scaling (7.1) because $\|\mathcal{A}\|$ is not available; see the source code for our approach. We apply SketchyCGAL with sketch size $R = \mathtt{n}$, using total storage $\Theta(\mathtt{n}^3)$. After rounding, a low- or medium-accuracy solution of (7.6) often provides a better permutation than a high-accuracy solution. Therefore, we applied the rounding step at iterations $2, 4, 8, 16, \ldots$ and tracked the quality of the best permutation attained on the solution path. We stopped after (the first of) 10^6 iterations or 48 hours.

The results of this experiment appear in Figure 7.5. We compare against the best value reported by Bravo Ferreira et al. in [26, Tables 4 and 6] for their CSDP method with clique size $k \in \{2, 3, 4\}$. We also include the results that [26] lists for the PATH method [116].

SketchyCGAL allows us to solve a tighter SDP relaxation of QAP than the other methods (CSDP, PATH). As a consequence, we obtain significantly smaller gaps for most instances. In fact, SketchyCGAL even recovers the exact solution for some of the instances in QAPLIB!

8. Related work.

There is a large body of literature on numerical methods for solving SDPs. Most of these algorithms adopt standard techniques from optimization. This section describes the major approaches, but an exhaustive discussion is beyond the scope of this work. See [75] for a recent survey.

8.1. Interior point method (IPM). The first practical IPM algorithms for linear programming were developed in 1984 by Karmarkar [62]. Soon after, they were extended to SDPs independently by Nesterov & Nemirovski [83, 84], Alizadeh [3, 4], and Kamath & Karmarkar [60, 61]. IPMs are still widely used for solving SDPs to high precision. Software packages include SeDuMi [97], MoSeK [78], and SDPT3 [99].

Unfortunately, IPMs do not scale to large problems. Indeed, an IPM formulates a sequence of unconstrained barrier problems, which are solved via Newton's method or its variants. This step requires storing and factoring large, dense matrices. As a result, a typical IPM for the SDP (2.2) requires $\Theta(n^3 + d^2n^2 + d^3)$ arithmetic operations per iteration and $\Theta(n^2 + dn + d^2)$

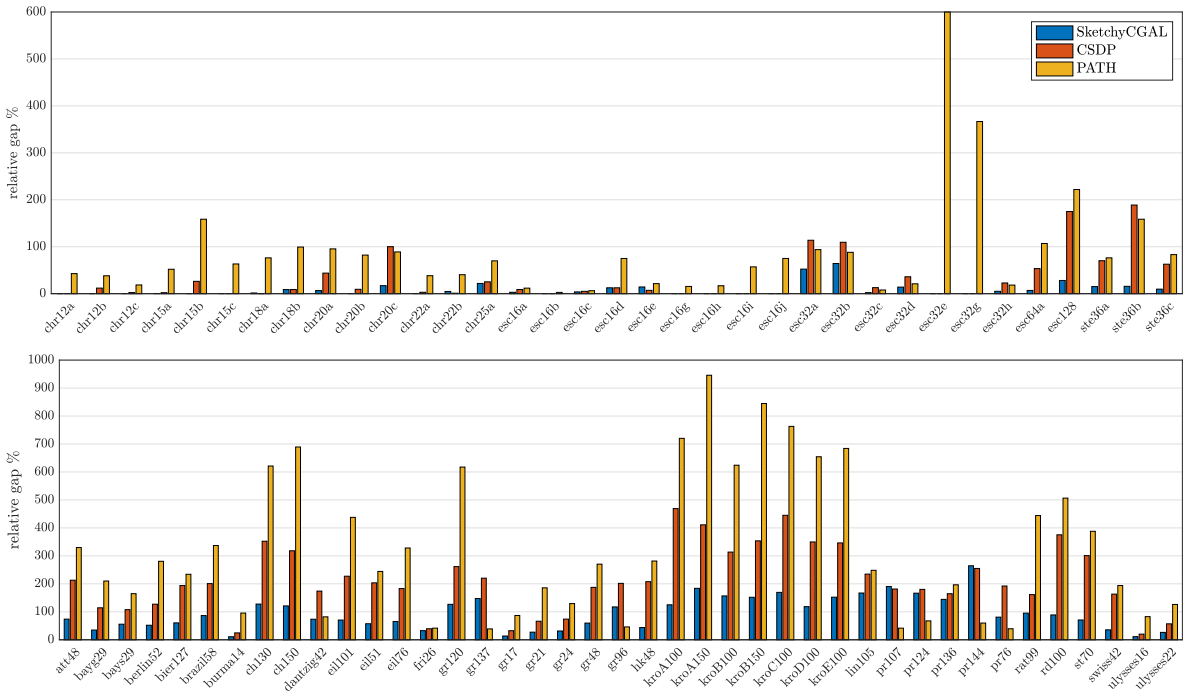


Figure 7.5. QAP relaxation: Solution quality. *Using SketchyCGAL, the CSDP method [26], and the PATH method [116], we solve SDP relaxations of QAP instances from QAPLIB [top] and TSPLIB [bottom]. The bars compare the cost of the computed solution, against the optimal value; shorter is better. See subsection 7.5.5.*

memory [6]. As d and n grow, time and storage costs become prohibitive, even for moderately sized problems, such as $n, d \gtrapprox 10\,000$. For more details, see [64, 9, 81, 19].

8.2. Augmented Lagrangian methods. Several effective SDP solvers are based on the augmented Lagrangian (AL) paradigm. These methods differ in the ways that the constraints are formulated and the subproblems are solved. See [29, 106, 118, 107] for examples.

Although AL techniques are often called "first-order methods," some of these algorithms use second-order information. In particular, Zhao et al. [118] employ a semi-smooth Newton method and the conjugate gradient method to solve the subproblems. Their method is enhanced and implemented in the software package SDPNAL+ [108].

Classical AL methods for SDPs typically require storage $\Omega(n^2)$. To address this weakness, researchers have combined the AL framework with CGM [45, 73, 95, 110] and with nonconvex matrix factorization [27, 28, 67, 93]. We explain some of these ideas below.

8.3. First-order methods. Next, we summarize the first-order methods for large SDPs. We focus on *projection-free* algorithms, which do not require full SVD computations.

8.3.1. Bundle methods. One of the earliest first-order SDP algorithms is the spectral bundle method developed by Helmberg and coauthors [53, 52, 51]. These algorithms build a piecewise affine model for the optimization problem using eigenvectors of the dual slack matrix $\boldsymbol{C} - \mathcal{A}\boldsymbol{y}^*$. At present, these methods lack a complete theoretical analysis.

8.3.2. Conditional gradient method (CGM) and friends. CGM solves a smooth convex minimization problem by repeatedly optimizing a linear approximation for the objective function over the constraint set. This approach was introduced by Frank & Wolfe [42] for optimization on polytopes; see also [71, 58]. Much later, Clarkson [35] developed a new analysis for optimization on the probability simplex Δ_n, and Hazan [50] studied optimization on the spectahedron $\boldsymbol{\Delta}_n$. Jaggi [57] generalized CGM to all compact, convex constraint sets.

The standard CGM algorithm does not apply to the model problem (2.2) because of the affine constraint $\mathcal{A}\boldsymbol{X} = \boldsymbol{b}$. Several variants [45, 73, 95, 111] of CGM can handle affine constraints. In particular, CGAL [110] does so by applying CGM to an AL formulation. We have chosen to extend CGAL because of its strong empirical performance; see [110, Sec. 4-5].

8.3.3. Primal-dual subgradient methods. Several authors have developed primal-dual subgradient algorithms [82, 113] that can be applied to SDPs. These methods perform subgradient ascent on the dual problem; the cost of each iteration is dominated by an eigenvector computation. Nesterov [82] constructs primal iterates by proximal mapping. The algorithm in Yurtsever et al. [113] constructs primal iterates by an averaging technique, similar to the updates in CGM. Bach [12] has described a relationship between CGM and dual subgradient ascent. In fact, CGAL and related methods are types of primal-dual averaging [114, 110].

8.4. Primal-dual games and online learning methods. Motivated by applications in theoretical computer science, researchers in this field have also devised SDP algorithms.

This approach first reduces SDP optimization to a sequence of feasibility problems. By standard duality techniques, each feasibility problem admits a saddle-point formulation:

$$\max_{\boldsymbol{X} \in \alpha \boldsymbol{\Delta}_n} \min_{\boldsymbol{p} \in \Delta_d} \quad \langle \boldsymbol{p}, \mathcal{A}\boldsymbol{X} - \boldsymbol{b} \rangle, \tag{8.1}$$

where Δ_d denotes the $(d-1)$-dimensional probability simplex. The dual of (8.1) is an eigenvalue optimization problem, which is important in its own right.

Most theoretical approaches to (8.1) are based on the matrix multiplicative weight (MMW) method [103, 10], which is nothing but the mirror-prox algorithm with the entropy mirror map [80]. The basic idea is to perform gradient descent in a dual space, using the matrix exponential map to transfer information back to the primal space. To make this approach more scalable, researchers have proposed linearization, random projection, and sparsification techniques to approximate the matrix exponential; see [11, 7, 88, 43, 44, 8, 14, 69].

This line of work has culminated in a simple rank-one MMW method [32], which uses the stochastic Lanczos method to approximate the matrix exponential. Even so, the reduction to a sequence of feasibility problems makes this technique impractical for solving general SDPs.

We are aware of only one experimental evaluation of the MMW idea, which appears in the randomized mirror-prox paper of Baes et al. [14].

8.5. Storage considerations. Almost all provably correct SDP algorithms store and operate on a full-dimensional matrix variable, so they are not suitable for large-scale SDPs.

Some primal-dual subgradient methods and CGM variants build an approximate solution as a convex combination of rank-one updates, so the rank of the solution does not exceed the number of iterations. This fact has led researchers to call these methods "storage-efficient," but this claim is misleading because the algorithms require many iterations to converge.

Recently, a subset of the authors has shown how to combine sketching with CGM to design a storage-optimal algorithm, SketchyCGM, for a class of low-rank matrix optimization problems [115]. We believe that SketchyCGM was the first provably correct storage-optimal method for these problems. SketchyCGAL was inspired by SketchyCGM.

In concurrent work with Ding [39], we developed another storage-optimal algorithm for standard-form SDPs. This method is based on a new *approximate complementarity principle*. Roughly, we can use a suboptimal dual point to approximate the range of the primal solution to the SDP. By compressing the primal problem to this subspace, we can solve the SDP with limited storage. This method, however, may fail on a set of SDPs with measure zero, and it has more limited guarantees than SketchyCGAL. A numerical evaluation is in process.

We have also noticed that the rank-one MMW method [32] can be combined with the Nyström sketch (Algorithm 5.1) to obtain a simple, storage-optimal algorithm for (8.1).

8.6. Nonconvex Burer–Monteiro methods. The most famous approach to low-storage semidefinite programming is the factorization heuristic proposed by Homer & Peinado [54] and extended by Burer & Monteiro (BM) [27]. The main idea is to rewrite the psd matrix variable $\boldsymbol{X} = \boldsymbol{F}\boldsymbol{F}^*$ in terms of a factor $\boldsymbol{F} \in \mathbb{R}^{n \times R}$, where the rank parameter $R \ll n$. We can reformulate the model problem (2.2) in terms of the new variable:

$$\text{minimize} \quad \langle \boldsymbol{C}, \boldsymbol{F}\boldsymbol{F}^* \rangle \quad \text{subject to} \quad \mathcal{A}(\boldsymbol{F}\boldsymbol{F}^*) = \boldsymbol{b}, \quad \boldsymbol{F}\boldsymbol{F}^* \in \alpha \boldsymbol{\Delta}_n. \tag{8.2}$$

This approach controls storage by sacrificing convexity and the associated guarantees. We can apply various nonlinear programming methods to optimize (8.2). AL methods are most commonly used [27, 28, 67, 93] because of their empirical success.

There has been an intense effort to establish theoretical guarantees for the BM factorization approach. It is clear that every solution to the SDP (2.2) of rank R or less is also a solution to the factorized problem (8.2). On the other hand, (8.2) may admit spurious local minima. Furthermore, because the formulation is nonconvex, we can only expect a numerical algorithm to locate a first- or second-order critical point.

Burer & Monteiro [28] developed the initial theory about (8.2): If $\boldsymbol{F}$ is a local minimum of (8.2) and if $R \geq \sqrt{2(d+1)}$, then $\boldsymbol{X}$ is either a global minimum or it is contained in the relative interior of a face of the feasible set of (2.2). Boumal et al. [23] obtained an improvement: If the constraint set of (8.2) is a smooth manifold and $R \geq \sqrt{2(d+1)}$ and $\boldsymbol{C}$ is generic, then each second-order critical point of (8.2) is a global optimum. A second-order critical point can be located using a Riemannian trust region method. See [21, 89, 34] for more extensions, but this theory requires opaque technical conditions that may not hold in practice.

The storage and arithmetic costs of solving (8.2) depend on the factorization rank R, as well as the optimization algorithm. Existing theoretical guarantees demand that $R = \Omega(\sqrt{d})$. In contrast, there are many applications where the SDP has a unique solution with *constant* rank. Thus, it is natural to ask whether we can reduce the rank parameter in the factorization. Waldspurger & Waters [105] resolved this question in the negative. Roughly, the BM formulation (8.2) can have spurious solutions unless $R = \Omega(\sqrt{d})$, and the bad problem instances form a set of positive measure. As a consequence, the Burer–Monteiro approach cannot support provably correct algorithms with storage costs better than $\Omega(n\sqrt{d})$. See Appendix F.2 for numerical evidence.

The most popular research software based on BM factorization is Manopt [22]. Solvers in Manopt implement manifold optimization algorithms, including Riemannian gradient and Riemannian trust region methods [1]. Consequently, Manopt is limited to problems where the factorized formulation (8.2) defines a smooth manifold.

8.7. Conclusion. We have presented a practical, new approach for solving standard-form SDPs at scale. Our algorithm combines a primal-dual optimization method with randomized linear algebra techniques to achieve unprecedented guarantees. We hope that our ideas lead to further algorithmic advances and support new applications of semidefinite programming.

Appendix A. Analysis of the CGAL Algorithm.

This section contains a complete analysis of the convergence of the CGAL algorithm and its arithmetic costs. For simplicity, we have specialized this presentation to the model problem that is the focus of this paper; many extensions are possible. The convergence results here are adapted from the initial paper [110] on the CGAL algorithm. The analysis of the arithmetic cost for the model problem is new.

A.1. The model problem. We focus on solving the optimization template

$$\text{minimize} \quad \langle \boldsymbol{C}, \boldsymbol{X} \rangle \quad \text{subject to} \quad \mathcal{A}\boldsymbol{X} = \boldsymbol{b}, \quad \boldsymbol{X} \in \alpha\boldsymbol{\Delta}_n. \tag{A.1}$$

The constraint set $\alpha\boldsymbol{\Delta}_n$ consists of $n \times n$ psd matrices with trace α. The objective function depends on a matrix $\boldsymbol{C} \in \mathbb{S}_n$. The linear constraints are determined by the linear map $\mathcal{A} : \mathbb{S}_n \to \mathbb{R}^d$ and the right-hand side vector $\boldsymbol{b} \in \mathbb{R}^d$.

A.2. Elements of Lagrangian duality. Introduce the Lagrangian function

$$L(\boldsymbol{X}; \boldsymbol{y}) := \langle \boldsymbol{C}, \boldsymbol{X} \rangle + \langle \boldsymbol{y}, \mathcal{A}\boldsymbol{X} - \boldsymbol{b} \rangle \quad \text{for } \boldsymbol{X} \in \alpha\boldsymbol{\Delta}_n \text{ and } \boldsymbol{y} \in \mathbb{R}^d. \tag{A.2}$$

We assume that the Lagrangian admits at least one saddle point $(\boldsymbol{X}_\star, \boldsymbol{y}_\star) \in \alpha\boldsymbol{\Delta}_n \times \mathbb{R}^d$:

$$L(\boldsymbol{X}_\star, \boldsymbol{y}) \le L(\boldsymbol{X}_\star, \boldsymbol{y}_\star) \le L(\boldsymbol{X}; \boldsymbol{y}_\star) \quad \text{for all } \boldsymbol{X} \in \alpha\boldsymbol{\Delta}_n \text{ and } \boldsymbol{y} \in \mathbb{R}^d. \tag{A.3}$$

This hypothesis is guaranteed under Slater's condition. Write $p_\star$ for the shared extremal value of the dual and primal problems:

$$\max_{\boldsymbol{y} \in \mathbb{R}^d} \min_{\boldsymbol{X} \in \alpha\boldsymbol{\Delta}_n} L(\boldsymbol{X}, \boldsymbol{y}) = p_\star = \min_{\boldsymbol{X} \in \alpha\boldsymbol{\Delta}_n} \sup_{\boldsymbol{y} \in \mathbb{R}^d} L(\boldsymbol{X}, \boldsymbol{y}). \tag{A.4}$$

In particular, note that $p_\star = \langle \boldsymbol{C}, \boldsymbol{X}_\star \rangle$.

A.3. The CGAL iteration. Let $\beta_0 > 0$ be an initial smoothing parameter. Fix a schedule for the step size parameter η_t and the smoothing parameter β_t:

$$\eta_t := \frac{2}{t+1} \quad \text{and} \quad \beta_t := \beta_0 \sqrt{t+1} \quad \text{for } t = 1, 2, 3, \ldots. \tag{A.5}$$

Define the augmented Lagrangian L_t with smoothing parameter β_t

$$L_t(\boldsymbol{X}; \boldsymbol{y}) := \langle \boldsymbol{C}, \boldsymbol{X} \rangle + \langle \boldsymbol{y}, \mathcal{A}\boldsymbol{X} - \boldsymbol{b} \rangle + \frac{1}{2}\beta_t \|\mathcal{A}\boldsymbol{X} - \boldsymbol{b}\|^2. \tag{A.6}$$

The CGAL algorithm solves the model problem (A.1) by alternating between primal and dual update steps on (A.6), while increasing the smoothing parameter.

Fix the initial iterates

$$\boldsymbol{X}_1 = \mathbf{0} \in \mathbb{S}_n \quad \text{and} \quad \boldsymbol{y}_1 = \mathbf{0} \in \mathbb{R}^d. \tag{A.7}$$

At each iteration $t = 1, 2, 3, \dots$, we implicitly compute the partial derivative

$$\boldsymbol{D}_t := \partial_{\boldsymbol{X}} L_t(\boldsymbol{X}_t; \boldsymbol{y}_t) = \boldsymbol{C} + \mathcal{A}^*\big(\boldsymbol{y}_t + \beta_t(\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b})\big). \tag{A.8}$$

Then we identify a primal update direction $\boldsymbol{H}_t \in \alpha\boldsymbol{\Delta}_n$ that satisfies

$$\langle \boldsymbol{D}_t, \boldsymbol{H}_t\rangle \le \min_{\boldsymbol{H}\in\alpha\boldsymbol{\Delta}_n} \langle \boldsymbol{D}_t, \boldsymbol{H}\rangle + \frac{\alpha\beta_0}{\beta_t}\|\boldsymbol{D}_t\|. \tag{A.9}$$

In other words, the smoothing parameter β_t also controls the amount of inexactness we are willing to tolerate in the linear minimization at iteration t. We construct the next primal iterate via the rule

$$\boldsymbol{X}_{t+1} = \boldsymbol{X}_t + \eta_t(\boldsymbol{H}_t - \boldsymbol{X}_t) \in \alpha\boldsymbol{\Delta}_n. \tag{A.10}$$

The dual update takes the form

$$\boldsymbol{y}_{t+1} = \boldsymbol{y}_t + \gamma_t(\mathcal{A}\boldsymbol{X}_{t+1} - \boldsymbol{b}). \tag{A.11}$$

We select the largest dual step size parameter $0 \le \gamma_t \le \beta_0$ that satisfies the condition

$$\gamma_t\|\mathcal{A}\boldsymbol{X}_{t+1} - \boldsymbol{b}\|^2 \le \beta_t\eta_t^2\alpha^2\|\mathcal{A}\|^2. \tag{A.12}$$

The latter inequality always holds when $\gamma_t = 0$. We will also choose the dual step size to maintain a bounded travel condition:

$$\|\boldsymbol{y}_t\| \le K. \tag{A.13}$$

If the bounded travel condition holds at iteration $t-1$, then the choice $\gamma_t = 0$ ensures that it holds at iteration t. In practice, it is not necessary to enforce (A.13). The iteration continues until it reaches a maximum iteration count $T_{\max}$.

A.4. Solution quality. We can evaluate the quality of a given primal iterate $\boldsymbol{X}_t$ with the information we have at hand. For simplicity, let us assume that we have solved the linear minimization (A.9) exactly at iteration t. That is, there is no error depending on $\|\boldsymbol{D}_t\|$. We can make this calculation periodically (say, for $t = 2^i$ for $i \in \mathbb{N}$) to decide whether to stop.

First, note that we have instant access to the infeasibility: $\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}$. Second, we can bound the suboptimality of the primal objective value. Introduce the surrogate duality gap

$$g_t(\boldsymbol{X}) := \max_{\boldsymbol{H}\in\alpha\boldsymbol{\Delta}_n} \langle \boldsymbol{D}_t, \boldsymbol{X} - \boldsymbol{H}\rangle.$$

The quantity $g_t(\boldsymbol{X}_t)$ can be determined from the solution to (A.9). Then

$$\langle \boldsymbol{C}, \boldsymbol{X}_t\rangle - p_\star \le g_t(\boldsymbol{X}_t) - \langle \boldsymbol{y}_t, \mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}\rangle.$$

See Appendix A.7 for discussion and a more refined estimate for the suboptimality.

Algorithm A.1 CGAL for the model problem (A.1)

Input: Problem data for (A.1) implemented via the primitives (2.4); number T of iterations
Output: Approximate solution $\boldsymbol{X}_T$ to (2.2)
Recommendation: Set $T \approx \varepsilon^{-1}$ to achieve ε-optimal solution (A.17)

function CGAL(T)
  Scale problem data (subsection 7.1.2) ▷ **[opt]** Recommended!
  $\beta_0 \leftarrow 1$ and $K \leftarrow +\infty$ ▷ Default parameters
  $\boldsymbol{X} \leftarrow \mathbf{0}_{n\times n}$ and $\boldsymbol{y} \leftarrow \mathbf{0}_d$
  for $t \leftarrow 1, 2, 3, \ldots, T$ **do**
    $\beta \leftarrow \beta_0\sqrt{t+1}$ and $\eta \leftarrow 2/(t+1)$
    $(\xi, \boldsymbol{v}) \leftarrow \mathsf{ApproxMinEvec}(\boldsymbol{C} + \mathcal{A}^*(\boldsymbol{y} + \beta(\mathcal{A}\boldsymbol{X} - \boldsymbol{b})); q_t)$
      ▷ Use primitives (2.4)❶❷!
      ▷ With Algorithm 4.1, set $q_t = 8t^{1/2}\log n$
      ▷ With Algorithm 4.2, set $q_t = t^{1/4}\log n$
    $\boldsymbol{X} \leftarrow (1-\eta)\,\boldsymbol{X} + \eta\,(\alpha\,\boldsymbol{v}\boldsymbol{v}^*)$
    $\boldsymbol{y} \leftarrow \boldsymbol{y} + \gamma(\mathcal{A}\boldsymbol{X} - \boldsymbol{b})$ ▷ Step size γ satisfies (A.12) and (A.13)

A.5. Theoretical analysis of CGAL. We develop two results on the behavior of CGAL. The first concerns its convergence to optimality, and the second concerns the computational resource usage.

A.5.1. Convergence. The first result demonstrates that the CGAL algorithm always converges to a primal optimal solution of the model problem (A.1). This result is adapted from [110]; a complete proof appears below in Appendix A.6.

Theorem A.1 (CGAL: Convergence). *Define*

$$E := 6\alpha^2\|\mathcal{A}\|^2 + \alpha(\|\boldsymbol{C}\| + K\|\mathcal{A}\|).$$

The primal iterates $\{\boldsymbol{X}_t : t = 2, 3, 4, \ldots\}$ *generated by the* CGAL *iteration satisfy the* a priori *bounds*

$$\langle \boldsymbol{C}, \boldsymbol{X}_t\rangle - p_\star \le \frac{2\beta_0 E}{\sqrt{t}} + K\cdot\|\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}\|; \tag{A.14}$$

$$-\|\boldsymbol{y}_\star\|\cdot\|\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}\| \le \langle \boldsymbol{C}, \boldsymbol{X}_t\rangle - p_\star; \tag{A.15}$$

$$\|\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}\| \le \frac{2\beta_0^{-1}(K + \|\boldsymbol{y}_\star\|) + 2\sqrt{E}}{\sqrt{t}}. \tag{A.16}$$

The a priori *bounds ensure that*

$$\|\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}\| \le \varepsilon \quad \text{and} \quad |\langle \boldsymbol{C}, \boldsymbol{X}_t\rangle - p_\star| \le \varepsilon \tag{A.17}$$

within $O(\varepsilon^{-2})$ *iterations. The big-O suppresses constants that depend on the problem data* $(\alpha, \|\mathcal{A}\|, \|\boldsymbol{C}\|)$ *and the algorithm parameters* β_0 *and* K.

A.5.2. Problem scaling. Theorem A.1 indicates that it is valuable to scale the model problem (A.1) so that $\alpha = \|\boldsymbol{C}\| = \|\mathcal{A}\| = 1$. In this case, a good choice for the smoothing parameter is $\beta_0 = 1$. Nevertheless, the algorithm converges, regardless of the scaling and regardless of the parameter choices β_0 and K. We use a slightly different scaling in practice; see subsection 7.1.2.

A.6. Proof of Theorem A.1. In this section, we establish the convergence guarantee stated in Theorem A.1.

A.6.1. Analysis of the primal update. The first steps in the proof address the role of the primal update rule (A.10). The arguments parallels the standard convergence analysis [57] of CGM, applied to the function

$$f_t(\boldsymbol{X}) := L_t(\boldsymbol{X}; \boldsymbol{y}_t) = \langle \boldsymbol{C}, \boldsymbol{X} \rangle + \langle \boldsymbol{y}_t, \mathcal{A}\boldsymbol{X} - \boldsymbol{b} \rangle + \frac{1}{2}\beta_t \|\mathcal{A}\boldsymbol{X} - \boldsymbol{b}\|^2. \tag{A.18}$$

Observe that the gradient $\nabla f_t(\boldsymbol{X}_t)$ coincides with the partial derivative (A.8).

To begin, we exploit the smoothness of f_t to control the change in its value at adjacent primal iterates. The function f_t is convex on $\mathbb{S}_n$, and its gradient has Lipschitz constant $\beta_t\|\mathcal{A}\|^2$, so

$$f_t(\boldsymbol{X}_+) - f_t(\boldsymbol{X}) \le \langle \nabla f_t(\boldsymbol{X}), \boldsymbol{X}_+ - \boldsymbol{X} \rangle + \frac{1}{2}\beta_t\|\mathcal{A}\|^2\|\boldsymbol{X}_+ - \boldsymbol{X}\|_{\mathrm{F}}^2 \quad \text{for } \boldsymbol{X}, \boldsymbol{X}_+ \in \mathbb{S}_n.$$

In particular, with $\boldsymbol{X}_+ = \boldsymbol{X}_{t+1}$ and $\boldsymbol{X} = \boldsymbol{X}_t$, we obtain

$$\begin{aligned} f_t(\boldsymbol{X}_{t+1}) &\le f_t(\boldsymbol{X}_t) + \langle \boldsymbol{D}_t, \boldsymbol{X}_{t+1} - \boldsymbol{X}_t \rangle + \frac{1}{2}\beta_t\|\mathcal{A}\|^2\|\boldsymbol{X}_{t+1} - \boldsymbol{X}_t\|_{\mathrm{F}}^2 \\ &= f_t(\boldsymbol{X}_t) + \eta_t\langle \boldsymbol{D}_t, \boldsymbol{H}_t - \boldsymbol{X}_t \rangle + \frac{1}{2}\beta_t\eta_t^2\|\mathcal{A}\|^2\|\boldsymbol{H}_t - \boldsymbol{X}_t\|_{\mathrm{F}}^2 \\ &\le f_t(\boldsymbol{X}_t) + \eta_t\langle \boldsymbol{D}_t, \boldsymbol{H}_t - \boldsymbol{X}_t \rangle + \beta_t\eta_t^2\alpha^2\|\mathcal{A}\|^2. \end{aligned} \tag{A.19}$$

The second identity follows from the update rule (A.10). The bound on the last term depends on the fact that the constraint set $\alpha\boldsymbol{\Delta}_n$ has Frobenius-norm diameter $\alpha\sqrt{2}$.

Next, we use the construction of the primal update to control the linear term in the last display. The update direction $\boldsymbol{H}_t$ satisfies the inequality (A.9), so

$$\begin{aligned} \langle \boldsymbol{D}_t, \boldsymbol{H}_t - \boldsymbol{X}_t \rangle &\le \min_{\boldsymbol{H} \in \alpha\boldsymbol{\Delta}_n} \langle \boldsymbol{D}_t, \boldsymbol{H} - \boldsymbol{X}_t \rangle + \frac{\alpha\beta_0}{\beta_t}\|\boldsymbol{D}_t\| \\ &\le \langle \boldsymbol{D}_t, \boldsymbol{X}_\star - \boldsymbol{X}_t \rangle + \frac{\alpha\beta_0}{\beta_t}\|\boldsymbol{D}_t\|. \end{aligned} \tag{A.20}$$

The second inequality depends on the fact that $\boldsymbol{X}_\star \in \alpha\boldsymbol{\Delta}_n$.

We can use the explicit formula (A.8) for the derivative $\boldsymbol{D}_t$ to control the two terms in (A.20). First,

$$\begin{aligned} \langle \boldsymbol{D}_t, \boldsymbol{X}_\star - \boldsymbol{X}_t \rangle &= \langle \boldsymbol{C} + \mathcal{A}^*\big(\boldsymbol{y}_t + \beta_t(\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b})\big), \boldsymbol{X}_\star - \boldsymbol{X}_t \rangle \\ &= \langle \boldsymbol{C}, \boldsymbol{X}_\star - \boldsymbol{X}_t \rangle - \langle \boldsymbol{y}_t, \mathcal{A}\boldsymbol{X}_t - \boldsymbol{b} \rangle - \beta_t\|\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}\|^2 \\ &= \langle \boldsymbol{C}, \boldsymbol{X}_\star \rangle - f_t(\boldsymbol{X}_t) - \frac{1}{2}\beta_t\|\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}\|^2. \end{aligned} \tag{A.21}$$

We have invoked the definition of the adjoint $\mathcal{A}^*$ and the fact that $\mathcal{A}\boldsymbol{X}_\star = \boldsymbol{b}$. Last, we used the definition (A.18) to identify the quantity $f_t(\boldsymbol{X}_t)$. Second,

$$\begin{aligned}\|\boldsymbol{D}_t\| &= \|\boldsymbol{C} + \mathcal{A}^*\big(\boldsymbol{y}_t + \beta_t(\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b})\big)\| \\ &\le \|\boldsymbol{C}\| + K\|\mathcal{A}\| + \beta_t\|\mathcal{A}\|\|\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}\|.\end{aligned} \tag{A.22}$$

We have used the assumption that $\boldsymbol{y}_t$ satisfies the bounded travel condition (A.13).

Combine the last three displays to obtain the estimate

$$\begin{aligned}\langle \boldsymbol{D}_t, \boldsymbol{H}_t - \boldsymbol{X}_t\rangle \le \langle \boldsymbol{C}, \boldsymbol{X}_\star\rangle - f_t(\boldsymbol{X}_t) \\ + \Big(\alpha\beta_0\|\mathcal{A}\|\|\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}\| - \frac{1}{2}\beta_t\|\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}\|^2\Big) + \frac{\alpha\beta_0}{\beta_t}(\|\boldsymbol{C}\| + K\|\mathcal{A}\|).\end{aligned} \tag{A.23}$$

We can now control the decrease in the function f_t between adjacent primal iterates. Combine the displays (A.19) and (A.23) to arrive at the bound

$$\begin{aligned}&f_t(\boldsymbol{X}_{t+1}) \le (1-\eta_t)f_t(\boldsymbol{X}_t) + \eta_t\langle \boldsymbol{C}, \boldsymbol{X}_\star\rangle \\ &+ \Big(\eta_t\alpha\beta_0\|\mathcal{A}\|\|\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}\| - \frac{1}{2}\beta_t\eta_t\|\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}\|^2\Big) + \Big(\beta_t\eta_t^2\alpha^2\|\mathcal{A}\|^2 + \frac{\eta_t\alpha\beta_0}{\beta_t}(\|\boldsymbol{C}\| + K\|\mathcal{A}\|)\Big).\end{aligned}$$

Subtract $p_\star = \langle \boldsymbol{C}, \boldsymbol{X}_\star\rangle$ from both sides to arrive at

$$\begin{aligned}&f_t(\boldsymbol{X}_{t+1}) - p_\star \le (1-\eta_t)\Big(f_t(\boldsymbol{X}_t) - p_\star\Big) \\ &+ \Big(\eta_t\alpha\beta_0\|\mathcal{A}\|\|\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}\| - \frac{1}{2}\beta_t\eta_t\|\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}\|^2\Big) + \Big(\beta_t\eta_t^2\alpha^2\|\mathcal{A}\|^2 + \frac{\eta_t\alpha\beta_0}{\beta_t}(\|\boldsymbol{C}\| + K\|\mathcal{A}\|)\Big).\end{aligned}$$

Finally, use the definition (A.18) again to pass back to the augmented Lagrangian:

$$\begin{aligned}L_t(\boldsymbol{X}_{t+1}; \boldsymbol{y}_t) - p_\star \le (1-\eta_t)\Big(L_t(\boldsymbol{X}_t; \boldsymbol{y}_t) - p_\star\Big) \\ + \Big(\eta_t\alpha\beta_0\|\mathcal{A}\|\|\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}\| - \frac{1}{2}\beta_t\eta_t\|\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}\|^2\Big) \\ + \Big(\beta_t\eta_t^2\alpha^2\|\mathcal{A}\|^2 + \frac{\eta_t\alpha\beta_0}{\beta_t}(\|\boldsymbol{C}\| + K\|\mathcal{A}\|)\Big).\end{aligned} \tag{A.24}$$

This bound describes the evolution of the augmented Lagrangian as the primal iterate advances. But we still need to include the effects of increasing the smoothing parameter and updating the dual variable.

A.6.2. Analysis of the smoothing update. To continue, observe that updates to the smoothing parameter have a controlled impact on the augmented Lagrangian (A.6):

$$L_t(\boldsymbol{X}_t; \boldsymbol{y}_t) - L_{t-1}(\boldsymbol{X}_t; \boldsymbol{y}_t) = \frac{1}{2}(\beta_t - \beta_{t-1})\|\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}\|^2.$$

Add and subtract $L_{t-1}(\boldsymbol{X}_t; \boldsymbol{y}_t)$ in the large parenthesis in the first line of (A.24) and invoke the last identity to obtain

$$\begin{aligned} L_t(\boldsymbol{X}_{t+1}; \boldsymbol{y}_t) - p_\star &\le (1-\eta_t)\Big(L_{t-1}(\boldsymbol{X}_t; \boldsymbol{y}_t) - p_\star\Big) \\ &\quad + \Big(\eta_t\alpha\beta_0\|\mathcal{A}\|\|\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}\| + \frac{1}{2}\left[(1-\eta_t)(\beta_t - \beta_{t-1}) - \beta_t\eta_t\right]\|\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}\|^2\Big) \\ &\quad + \Big(\beta_t\eta_t^2\alpha^2\|\mathcal{A}\|^2 + \frac{\eta_t\alpha\beta_0}{\beta_t}(\|\boldsymbol{C}\| + K\|\mathcal{A}\|)\Big). \end{aligned} \tag{A.25}$$

The next step is to develop a uniform bound on the terms in the second line so that we can ignore the role of the feasibility gap $\|\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}\|$ in the subsequent calculations. The choice (A.5) of the parameters ensures that

$$(1-\eta_t)(\beta_t - \beta_{t-1}) - \beta_t\eta_t < \frac{-\beta_0^2}{\beta_t}.$$

Introduce this bound into the second line of (A.25) and maximize the resulting concave quadratic function to reach

$$\begin{aligned} &\eta_t\alpha\beta_0\|\mathcal{A}\|\|\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}\| + \frac{1}{2}\left[(1-\eta_t)(\beta_t - \beta_{t-1}) - \beta_t\eta_t\right]\|\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}\|^2 \\ &\qquad \le \eta_t\alpha\beta_0\|\mathcal{A}\|\|\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}\| - \frac{\beta_0^2}{2\beta_t}\|\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}\|^2 \le \beta_t\eta_t^2\alpha^2\|\mathcal{A}\|^2. \end{aligned}$$

Substitute the last display into (A.25) to determine that

$$\begin{aligned} L_t(\boldsymbol{X}_{t+1}; \boldsymbol{y}_t) - p_\star &\le (1-\eta_t)\Big(L_{t-1}(\boldsymbol{X}_t; \boldsymbol{y}_t) - p_\star\Big) \\ &\quad + \Big(2\beta_t\eta_t^2\alpha^2\|\mathcal{A}\|^2 + \frac{\eta_t\alpha\beta_0}{\beta_t}(\|\boldsymbol{C}\| + K\|\mathcal{A}\|)\Big). \end{aligned} \tag{A.26}$$

To develop a recursion, we need to assess how the left-hand side changes when we update the dual variable.

A.6.3. Analysis of the dual update. To incorporate the dual update, observe that

$$\begin{aligned} L_t(\boldsymbol{X}_{t+1}; \boldsymbol{y}_{t+1}) &= L_t(\boldsymbol{X}_{t+1}; \boldsymbol{y}_t) + \langle \boldsymbol{y}_{t+1} - \boldsymbol{y}_t,\, \mathcal{A}\boldsymbol{X}_{t+1} - \boldsymbol{b}\rangle \\ &= L_t(\boldsymbol{X}_{t+1}; \boldsymbol{y}_t) + \gamma_t\|\mathcal{A}\boldsymbol{X}_{t+1} - \boldsymbol{b}\|^2 \\ &\le L_t(\boldsymbol{X}_{t+1}; \boldsymbol{y}_t) + \beta_t\eta_t^2\alpha^2\|\mathcal{A}\|^2. \end{aligned}$$

The first relation is simply the definition (A.6) of the augmented Lagrangian, while the second relation depends on the dual update rule (A.11). The last step follows from the selection rule (A.12) for the dual step size parameter.

Introduce the latter display into (A.25) to discover that

$$\begin{aligned} L_t(\boldsymbol{X}_{t+1}; \boldsymbol{y}_{t+1}) - p_\star &\le (1-\eta_t)\Big(L_{t-1}(\boldsymbol{X}_t; \boldsymbol{y}_t) - p_\star\Big) \\ &\quad + \Big(3\beta_t\eta_t^2\alpha^2\|\mathcal{A}\|^2 + \frac{\eta_t\alpha\beta_0}{\beta_t}(\|\boldsymbol{C}\| + K\|\mathcal{A}\|)\Big). \end{aligned} \tag{A.27}$$

We have developed a recursion for the value of the augmented Lagrangian as the iterates and the smoothing parameter evolve.

A.6.4. Solving the recursion. Next, we solve the recursion (A.27). We assert that

$$\begin{aligned} L_t(\boldsymbol{X}_{t+1}; \boldsymbol{y}_{t+1}) - p_\star &< \frac{2\beta_0}{\sqrt{t+1}} \left[6\alpha^2 \|\mathcal{A}\|^2 + \alpha(\|\boldsymbol{C}\| + K\|\mathcal{A}\|)\right] \\ &=: \frac{2\beta_0 E}{\sqrt{t+1}} \quad \text{for } t = 1, 2, 3, \dots. \end{aligned} \tag{A.28}$$

For the case $t = 1$, the definition (A.5) ensures that $\eta_1 = 1$ and $\beta_1 = \beta_0\sqrt{2}$, so the bound (A.28) follows instantly from (A.27). When $t > 1$, an inductive argument using the recursion (A.27) and the bound (A.28) for $t - 1$ ensures that

$$\begin{aligned} L_t(\boldsymbol{X}_{t+1}; \boldsymbol{y}_{t+1}) - p_\star &\le \left[\frac{t-1}{t+1} \cdot \frac{2\beta_0}{\sqrt{t}} + \frac{2\beta_0}{(t+1)^{3/2}}\right] \left[6\alpha^2 \|\mathcal{A}\|^2 + \alpha(\|\boldsymbol{C}\| + K\|\mathcal{A}\|)\right] \\ &< \frac{2\beta_0}{\sqrt{t+1}} \left[6\alpha^2 \|\mathcal{A}\|^2 + \alpha(\|\boldsymbol{C}\| + K\|\mathcal{A}\|)\right]. \end{aligned}$$

We have introduced the stated values (A.5) of the step size and smoothing parameters. The induction proceeds, and we conclude that (A.28) is valid.

A.6.5. Bound for the suboptimality of the objective. We are prepared to develop an upper bound on the suboptimality of the objective of the model problem (A.1). The definition (A.6) of the augmented Lagrangian directly implies that

$$\begin{aligned} \langle \boldsymbol{C}, \boldsymbol{X}_{t+1} \rangle - p_\star = L_t(\boldsymbol{X}_{t+1}; \boldsymbol{y}_{t+1}) - p_\star - \langle \boldsymbol{y}_{t+1}, \mathcal{A}\boldsymbol{X}_{t+1} - \boldsymbol{b} \rangle \\ - \frac{1}{2}\beta_t \|\mathcal{A}\boldsymbol{X}_{t+1} - \boldsymbol{b}\|^2. \end{aligned} \tag{A.29}$$

Continuing from here,

$$\langle \boldsymbol{C}, \boldsymbol{X}_{t+1} \rangle - p_\star \le \frac{2\beta_0 E}{\sqrt{t+1}} + K \cdot \|\mathcal{A}\boldsymbol{X}_{t+1} - \boldsymbol{b}\|.$$

The first identity follows from definition (A.6) of the augmented Lagrangian. The bound relies on (A.28) and the Cauchy–Schwarz inequality. We have also used the bounded travel condition (A.13). This establishes (A.14).

A.6.6. Bound for the superoptimality of the objective. The CGAL iterates $\boldsymbol{X}_t$ are generally infeasible for (A.1), so they can be superoptimal. Nevertheless, we can easily control the superoptimality. By the saddle-point properties (A.3) and (A.4), the Lagrangian (A.2) satisfies

$$p_\star = \max_{\boldsymbol{y}} \min_{\boldsymbol{X} \in \alpha\boldsymbol{\Delta}_n} L(\boldsymbol{X}; \boldsymbol{y}) \le L(\boldsymbol{X}_{t+1}; \boldsymbol{y}_\star) = \langle \boldsymbol{C}, \boldsymbol{X}_{t+1} \rangle + \langle \boldsymbol{y}_\star, \mathcal{A}\boldsymbol{X}_{t+1} - \boldsymbol{b} \rangle. \tag{A.30}$$

Invoke the Cauchy–Schwarz inequality and rearrange to determine that

$$-\|\boldsymbol{y}_\star\| \cdot \|\mathcal{A}\boldsymbol{X}_{t+1} - \boldsymbol{b}\| \le \langle \boldsymbol{C}, \boldsymbol{X}_{t+1} \rangle - p_\star.$$

This establishes (A.15).

A.6.7. Bound for the infeasibility of the iterates. Next, we demonstrate that the iterates converge toward the feasible set of (A.1). Combine (A.29) and (A.30) and rearrange to see that

$$\langle \boldsymbol{y}_{t+1} - \boldsymbol{y}_\star,\, \mathcal{A}\boldsymbol{X}_{t+1} - \boldsymbol{b}\rangle \le L_t(\boldsymbol{X}_{t+1}; \boldsymbol{y}_{t+1}) - p_\star - \frac{1}{2}\beta_t \|\mathcal{A}\boldsymbol{X}_{t+1} - \boldsymbol{b}\|^2.$$

Bound the left-hand side below using Cauchy–Schwarz and the right-hand side above using (A.28):

$$-\|\boldsymbol{y}_{t+1} - \boldsymbol{y}_\star\| \cdot \|\mathcal{A}\boldsymbol{X}_{t+1} - \boldsymbol{b}\| \le \frac{2\beta_0 E}{\sqrt{t+1}} - \frac{1}{2}\beta_t \|\mathcal{A}\boldsymbol{X}_{t+1} - \boldsymbol{b}\|^2.$$

Solve this quadratic inequality to obtain the bound

$$\begin{aligned}\|\mathcal{A}\boldsymbol{X}_{t+1} - \boldsymbol{b}\| &\le \beta_t^{-1}\left(\|\boldsymbol{y}_{t+1} - \boldsymbol{y}_\star\| + \sqrt{\|\boldsymbol{y}_{t+1} - \boldsymbol{y}_\star\|^2 + \frac{4\beta_t\beta_0 E}{\sqrt{t+1}}}\right) \\ &\le \beta_t^{-1}\left(2\|\boldsymbol{y}_{t+1} - \boldsymbol{y}_\star\| + \sqrt{4\beta_0^2 E}\right) \\ &= \frac{2\beta_0^{-1}\|\boldsymbol{y}_{t+1} - \boldsymbol{y}_\star\| + 2\sqrt{E}}{\sqrt{t+1}}.\end{aligned}$$

The second inequality depends on the definition (A.5) of the smoothing parameter β_t and the subadditivity of the square root.

Finally, we control the dual error using the bounded travel condition (A.13):

$$\|\boldsymbol{y}_{t+1} - \boldsymbol{y}_\star\| \le \|\boldsymbol{y}_{t+1}\| + \|\boldsymbol{y}_\star\| \le K + \|\boldsymbol{y}_\star\|.$$

The last two displays yield

$$\|\mathcal{A}\boldsymbol{X}_{t+1} - \boldsymbol{b}\| \le \frac{2\beta_0^{-1}(K + \|\boldsymbol{y}_\star\|) + 2\sqrt{E}}{\sqrt{t+1}}.$$

This confirms (A.16).

A.7. Computable bounds for suboptimality. In this section, we assume that the linear minimization (A.9) is performed exactly at iteration t. Introduce the duality gap surrogate

$$g_t(\boldsymbol{X}) := \max_{\boldsymbol{H} \in \alpha\boldsymbol{\Delta}_n} \langle \nabla f_t(\boldsymbol{X}),\, \boldsymbol{X} - \boldsymbol{H}\rangle. \tag{A.31}$$

The function f_t is defined in (A.18). Note that the gap $g(\boldsymbol{X}_t)$ can be evaluated with the information we have at hand:

$$\begin{aligned} g_t(\boldsymbol{X}_t) &= \langle \boldsymbol{D}_t,\, \boldsymbol{X}_t\rangle - \langle \boldsymbol{D}_t,\, \boldsymbol{H}_t\rangle \\ &= \langle \boldsymbol{C},\, \boldsymbol{X}_t\rangle + \langle \boldsymbol{y}_t + \beta_t(\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}),\, \mathcal{A}\boldsymbol{X}_t\rangle - \langle \boldsymbol{D}_t,\, \boldsymbol{H}_t\rangle. \end{aligned} \tag{A.32}$$

The last term is just the value of the linear minimization (A.9)

The gap gives us computable bounds on the suboptimality of the current iterate $\boldsymbol{X}_t$. Indeed, the convexity of f_t implies that

$$f_t(\boldsymbol{X}_t) - f_t(\boldsymbol{X}_\star) \le \langle \nabla f_t(\boldsymbol{X}_t),\, \boldsymbol{X}_t - \boldsymbol{X}_\star\rangle \le g_t(\boldsymbol{X}_t).$$

Using the definition (A.18) and the fact that $\boldsymbol{X}_\star$ is feasible for (A.1), we find that

$$f_t(\boldsymbol{X}_t) - p_\star \leq f_t(\boldsymbol{X}_t) - f_t(\boldsymbol{X}_\star) \leq g_t(\boldsymbol{X}_t).$$

Invoking the definition (A.18) again and rearranging, we find that

$$\begin{aligned} \langle \boldsymbol{C}, \boldsymbol{X}_t \rangle - p_\star &\leq g_t(\boldsymbol{X}_t) - \langle \boldsymbol{y}_t, \mathcal{A}\boldsymbol{X}_t - \boldsymbol{b} \rangle - \frac{1}{2}\beta_t \|\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}\|^2 \\ &\leq g_t(\boldsymbol{X}_t) - \langle \boldsymbol{y}_t, \mathcal{A}\boldsymbol{X}_t - \boldsymbol{b} \rangle. \end{aligned} \tag{A.33}$$

In other words, the suboptimality of the primal iterate $\boldsymbol{X}_t$ is bounded in terms of the dual gap $g_t(\boldsymbol{X}_t)$, the feasibility gap $\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}$, and the dual variable $\boldsymbol{y}_t$.

Appendix B. Sketching and reconstruction of a positive-semidefinite matrix.

This section reviews and gives additional details about the Nyström sketch [49, 46, 72, 100]. This sketch tracks an evolving psd matrix and then reports a provably accurate low-rank approximation. The material on error estimation is new.

B.1. Sketching and updates. Consider a psd input matrix $\boldsymbol{X} \in \mathbb{S}_n$. Let R be a parameter that modulates the storage cost of the sketch and the quality of the matrix approximation.

To construct the sketch, we draw and fix a standard normal test matrix $\boldsymbol{\Omega} \in \mathbb{F}^{n \times R}$. The sketch of the matrix $\boldsymbol{X}$ takes the form

$$\boldsymbol{S} = \boldsymbol{X}\boldsymbol{\Omega} \in \mathbb{F}^{n \times R} \quad \text{and} \quad \tau = \operatorname{tr}(\boldsymbol{X}) \in \mathbb{R}. \tag{B.1}$$

The sketch supports linear rank-one updates to $\boldsymbol{X}$. Indeed, we can track the evolution

$$\begin{aligned} \boldsymbol{X} &\leftarrow (1-\eta)\,\boldsymbol{X} + \eta\,\boldsymbol{v}\boldsymbol{v}^* \qquad \text{for } \eta \in [0,1] \text{ and } \boldsymbol{v} \in \mathbb{F}^n \\ \text{via} \quad \boldsymbol{S} &\leftarrow (1-\eta)\,\boldsymbol{S} + \eta\,\boldsymbol{v}(\boldsymbol{v}^*\boldsymbol{\Omega}) \\ \text{and} \quad \tau &\leftarrow (1-\eta)\,\tau + \eta\,\|\boldsymbol{v}\|^2. \end{aligned} \tag{B.2}$$

The test matrix $\boldsymbol{\Omega}$ and the sketch $(\boldsymbol{S}, \tau)$ require storage of $2Rn + 1$ numbers in $\mathbb{F}$. The arithmetic cost of the linear update (5.2) to the sketch is $\Theta(Rn)$ numerical operations.

Remark B.1 (Structured random matrices). We can reduce storage costs by a factor of two by using a structured random matrix in place of $\boldsymbol{\Omega}$. For example, see [102, Sec. 3] or [98]. This modification requires additional care with implementation (e.g., use of sparse arithmetic or fast trigonometric transforms), but the improvement can be significant for very large problems.

B.2. The reconstruction process. Given the test matrix $\boldsymbol{\Omega}$ and the sketch $\boldsymbol{S} = \boldsymbol{X}\boldsymbol{\Omega}$, we can form a rank-$R$ approximation $\widehat{\boldsymbol{X}}$ of the matrix $\boldsymbol{X}$ contained in the sketch. This approximation is defined by the formula

$$\widehat{\boldsymbol{X}} := \boldsymbol{S}(\boldsymbol{\Omega}^*\boldsymbol{S})^\dagger \boldsymbol{S}^* = (\boldsymbol{X}\boldsymbol{\Omega})(\boldsymbol{\Omega}^*\boldsymbol{X}\boldsymbol{\Omega})^\dagger(\boldsymbol{X}\boldsymbol{\Omega})^*. \tag{B.3}$$

This reconstruction is called a *Nyström approximation*. We often truncate $\widehat{\boldsymbol{X}}$ by replacing it with its best rank-r approximation $[\![\widehat{\boldsymbol{X}}]\!]_r$ for some $r \leq R$.

See Algorithm B.1 for a numerically stable implementation of the Nyström reconstruction process (5.3), including error estimation steps. The algorithm takes $\mathcal{O}(R^2 n)$ numerical operations and $\Theta(Rn)$ storage.

Algorithm B.1 NystromSketch implementation (see Appendix B)

Input: Dimension n of input matrix, size R of sketch
Output: Rank-R approximation $\widehat{\boldsymbol{X}}$ of sketched matrix in factored form $\widehat{\boldsymbol{X}} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^*$, where $\boldsymbol{U} \in \mathbb{R}^{n\times R}$ has orthonormal columns and $\boldsymbol{\Lambda} \in \mathbb{R}^{R\times R}$ is nonnegative diagonal, and the Schatten 1-norm approximation errors $\mathrm{err}(\llbracket\widehat{\boldsymbol{X}}\rrbracket_r)$ for $1 \le r \le R$, as defined in (B.4)
Recommendation: Choose R as large as possible

function NystromSketch.Init(n, R)
  $\boldsymbol{\Omega} \leftarrow \mathtt{randn}(n, R)$ ▷ Draw and fix random test matrix
  $\boldsymbol{S} \leftarrow \mathtt{zeros}(n, R)$ and $\tau \leftarrow 0$ ▷ Form sketch of zero matrix

function NystromSketch.RankOneUpdate($\boldsymbol{v}$, η) ▷ Implements (5.2)
  $\boldsymbol{S} \leftarrow (1-\eta)\,\boldsymbol{S} + \eta\,\boldsymbol{v}(\boldsymbol{v}^*\boldsymbol{\Omega})$ ▷ Update sketch of matrix
  $\tau \leftarrow (1-\eta)\,\tau + \eta\,\|\boldsymbol{v}\|^2$ ▷ Update the trace

function NystromSketch.Reconstruct
  $\sigma \leftarrow \sqrt{n}\,\mathtt{eps}(\mathtt{norm}(\boldsymbol{S}))$ ▷ Compute a shift parameter
  $\boldsymbol{S} \leftarrow \boldsymbol{S} + \sigma\,\boldsymbol{\Omega}$ ▷ Implicitly form sketch of $\boldsymbol{X} + \sigma\,\mathbf{I}$
  $\boldsymbol{C} \leftarrow \mathtt{chol}(\boldsymbol{\Omega}^*\boldsymbol{S})$
  $(\boldsymbol{U}, \boldsymbol{\Sigma}, \sim) \leftarrow \mathtt{svd}(\boldsymbol{S}/\boldsymbol{C})$ ▷ Dense SVD
  $\boldsymbol{\Lambda} \leftarrow \max\{0, \boldsymbol{\Sigma}^2 - \sigma\,\mathbf{I}\}$ ▷ Remove shift
  $\mathbf{err} \leftarrow \tau - \mathtt{cumsum}(\mathtt{diag}(\boldsymbol{\Lambda}))$ ▷ Compute approximation errors

B.3. The approximation error. We can easily determine the exact Schatten 1-norm error in the truncated Nyström approximation:

$$\mathrm{err}(\llbracket\widehat{\boldsymbol{X}}\rrbracket_r) := \|\boldsymbol{X} - \llbracket\widehat{\boldsymbol{X}}\rrbracket_r\|_* = \tau - \mathrm{tr}(\llbracket\widehat{\boldsymbol{X}}\rrbracket_r) \quad \text{for each } r \le R. \tag{B.4}$$

Furthermore, we can use (B.4) to ascertain whether the unknown input matrix $\boldsymbol{X}$ is (almost) low-rank. Indeed, the best rank-r approximation of $\boldsymbol{X}$ satisfies

$$\|\boldsymbol{X} - \llbracket\boldsymbol{X}\rrbracket_r\|_* \le \mathrm{err}(\llbracket\widehat{\boldsymbol{X}}\rrbracket_r) \quad \text{for each } r \le R. \tag{B.5}$$

Thus, large drops in the function $r \mapsto \mathrm{err}(\llbracket\widehat{\boldsymbol{X}}\rrbracket_r)$ signal large drops in the eigenvalues of $\boldsymbol{X}$. See Appendix B.8 for the details.

B.4. Statistical properties of the Nyström sketch. The truncated Nyström approximation has a number of attractive statistical properties. For a fixed input matrix $\boldsymbol{X}$, the expected approximation error $\mathbb{E}_{\boldsymbol{\Omega}}\,\|\boldsymbol{X} - \llbracket\widehat{\boldsymbol{X}}\rrbracket_r\|$ is monotone decreasing in both the sketch size R and the truncation rank r. Furthermore, if $\mathrm{rank}(\boldsymbol{X}) = r$ for $r \le R$, then $\|\boldsymbol{X} - \llbracket\widehat{\boldsymbol{X}}\rrbracket_r\|_* = 0$ with probability one. We establish these results below in Appendix B.9.

B.5. A priori error bounds. The Nyström approximation $\widehat{\boldsymbol{X}}$ yields a provably good estimate for the matrix $\boldsymbol{X}$ contained in the sketch [100, Thms. 4.1].

Fact B.2 (Nyström sketch: Error bound). *Fix a psd matrix $\boldsymbol{X} \in \mathbb{S}_n$. Let $\boldsymbol{S} = \boldsymbol{X}\boldsymbol{\Omega}$ where $\boldsymbol{\Omega} \in \mathbb{F}^{n\times R}$ is standard normal. For each $r < R$, the Nyström approximation (B.3) satisfies*

$$\mathbb{E}_{\boldsymbol{\Omega}} \|\boldsymbol{X} - \widehat{\boldsymbol{X}}\|_* \leq \left(1 + \frac{r}{R-r-1}\right) \|\boldsymbol{X} - [\![\boldsymbol{X}]\!]_r\|_*. \tag{B.6}$$

If we replace $\widehat{\boldsymbol{X}}$ with the rank-r truncation $[\![\widehat{\boldsymbol{X}}]\!]_r$, the error bound (B.6) remains valid. Similar results hold with high probability.

The truncated Nyström approximations satisfy a stronger error bound when the input matrix $\boldsymbol{X}$ exhibits spectral decay.

Fact B.3 (Nyström sketch: Error bound II). *Fix a psd matrix $\boldsymbol{X} \in \mathbb{S}_n$. Let $\boldsymbol{S} = \boldsymbol{X}\boldsymbol{\Omega}$ where $\boldsymbol{\Omega} \in \mathbb{F}^{n\times R}$ is standard normal. For each $r < R$, the Nyström approximation (B.3) satisfies*

$$\mathbb{E}_{\boldsymbol{\Omega}} \|\boldsymbol{X} - [\![\widehat{\boldsymbol{X}}]\!]_r\|_* \leq \|\boldsymbol{X} - [\![\boldsymbol{X}]\!]_r\|_* + \left(1 + \frac{r}{R-r-1}\right) \|\boldsymbol{X} - [\![\boldsymbol{X}]\!]_r\|_*. \tag{B.7}$$

If we replace $\widehat{\boldsymbol{X}}$ with the rank-r truncation $[\![\widehat{\boldsymbol{X}}]\!]_r$, the error bound (B.7) remains valid. Similar results hold with high probability.

Proof. Combine the proofs of [100, Thm. 4.2] and [49, Thm. 9.3]. ∎

B.6. Discussion. In practice, it is best to minimize the error attributable to sketching. To that end, we recommend choosing the sketch size parameter R as large as possible, given resource constraints, so that we can obtain the highest-quality Nyström approximation.

In some problems, e.g., MaxCut with eigenvector rounding, the desired rank r of the truncation is known in advance. In this case, Fact 5.2 offers guidance about how to select R to achieve a specific error tolerance $(1+\zeta)$ in (2.3). For example, when $5r + 1 \leq R$, the expected Schatten 1-norm error in the rank-r approximation $[\![\widehat{\boldsymbol{X}}]\!]_r$ is at most $1.25\times$ the error in the best rank-r approximation of $\boldsymbol{X}$.

When the input matrix $\boldsymbol{X}$ has decaying eigenvalues, the error in the truncated approximation may be far smaller than Fact 5.2 predicts; see [100, Thm. 4.2]. This happy situation is typical when $\boldsymbol{X}$ is generated by the CGAL iteration.

B.7. Representation of the truncated Nyström approximation. The key tool in the analysis is a simple representation for the truncated approximation. These facts are extracted from [100, Supp.].

Let $\boldsymbol{X} \in \mathbb{S}_n$ be a fixed psd matrix, and let $\boldsymbol{\Omega} \in \mathbb{R}^{n\times R}$ be an arbitrary test matrix. Let $\boldsymbol{P} \in \mathbb{S}_n$ be the orthoprojector onto the range of $\boldsymbol{X}^{1/2}\boldsymbol{\Omega}$. Then we can write the Nyström approximation (B.3) as

$$\widehat{\boldsymbol{X}} = \boldsymbol{X}^{1/2}\boldsymbol{P}\boldsymbol{X}^{1/2}.$$

For each $r \leq R$, define $\boldsymbol{P}_r$ to be the orthoprojector onto the co-range of the matrix $[\![\boldsymbol{X}^{1/2}\boldsymbol{P}]\!]_r$. By construction, $[\![\boldsymbol{X}^{1/2}\boldsymbol{P}]\!]_r = \boldsymbol{X}^{1/2}\boldsymbol{P}_r$. As a consequence,

$$[\![\widehat{\boldsymbol{X}}]\!]_r = ([\![\boldsymbol{X}^{1/2}\boldsymbol{P}]\!]_r)([\![\boldsymbol{X}^{1/2}\boldsymbol{P}]\!]_r)^* = \boldsymbol{X}^{1/2}\boldsymbol{P}_r\boldsymbol{X}^{1/2}.$$

These results allow us to relate the truncated approximations to each other:

B.8. Approximation errors: Analysis. We may now obtain explicit formulas for the error in each Nyström approximation. For $r \leq R$, note that

$$\boldsymbol{X} - [\![\widehat{\boldsymbol{X}}]\!]_r = \boldsymbol{X}^{1/2}(\mathbf{I} - \boldsymbol{P}_r)\boldsymbol{X}^{1/2} \succcurlyeq \mathbf{0}.$$

It follows immediately that

$$\operatorname{err}([\![\widehat{\boldsymbol{X}}]\!]_r) = \|\boldsymbol{X} - [\![\widehat{\boldsymbol{X}}]\!]_r\|_* = \operatorname{tr}(\boldsymbol{X} - [\![\widehat{\boldsymbol{X}}]\!]_r) = \operatorname{tr}(\boldsymbol{X}) - \operatorname{tr}([\![\widehat{\boldsymbol{X}}]\!]_r).$$

This is the relation (B.4).

Assume that $r \leq r'$. Since $\operatorname{tr}([\![\widehat{\boldsymbol{X}}]\!]_r) \leq \operatorname{tr}([\![\widehat{\boldsymbol{X}}]\!]_{r'})$, we have the bound

$$\operatorname{err}([\![\widehat{\boldsymbol{X}}]\!]_r) \geq \operatorname{err}([\![\widehat{\boldsymbol{X}}]\!]_{r'}) \quad \text{for } r \leq r'.$$

In other words, for fixed sketch size R, the error in the truncated Nyström approximation is monotone decreasing in the approximation rank.

To obtain (B.5), observe that

$$\|\boldsymbol{X} - [\![\boldsymbol{X}]\!]_r\|_* \leq \|\boldsymbol{X} - [\![\widehat{\boldsymbol{X}}]\!]_r\|_* = \operatorname{err}([\![\widehat{\boldsymbol{X}}]\!]_r).$$

The inequality holds because $[\![\boldsymbol{X}]\!]_r$ is a best rank-r approximation of $\boldsymbol{X}$ in Schatten 1-norm, while $[\![\widehat{\boldsymbol{X}}]\!]_r$ is another rank-r matrix.

B.9. Statistical properties: Analysis. Next, let us verify the statistical properties of the error. In this section, the test matrix $\boldsymbol{\Omega} \in \mathbb{F}^{n \times R}$ is standard normal.

Assuming that $\operatorname{rank}(\boldsymbol{X}) = r \leq R$, let us prove that $\|\boldsymbol{X} - [\![\boldsymbol{X}]\!]_r\|_* = 0$ with probability one. To that end, we observe

$$\operatorname{range}(\boldsymbol{P}) = \operatorname{range}(\boldsymbol{X}^{1/2}\boldsymbol{\Omega}) = \operatorname{range}(\boldsymbol{X}^{1/2}) \quad \text{with probability one.}$$

It follows that

$$\widehat{\boldsymbol{X}} = \boldsymbol{X}^{1/2}\boldsymbol{P}\boldsymbol{X}^{1/2} = \boldsymbol{X} \quad \text{with probability one.}$$

Moreover, $\operatorname{rank}(\boldsymbol{X}^{1/2}\boldsymbol{P}) = r$ with probability one. Conditional on this event,

$$[\![\widehat{\boldsymbol{X}}]\!]_r = \boldsymbol{X}^{1/2}\boldsymbol{P}_r\boldsymbol{X}^{1/2} = ([\![\boldsymbol{X}^{1/2}\boldsymbol{P}]\!]_r)([\![\boldsymbol{X}^{1/2}\boldsymbol{P}]\!]_r)^* = (\boldsymbol{X}^{1/2}\boldsymbol{P})(\boldsymbol{X}^{1/2}\boldsymbol{P})^* = \boldsymbol{X}.$$

This is the stated result.

Next, we show that the expected error in the truncated Nyström approximation is monotone decreasing with respect to the sketch size R. Fix the truncation rank r. Let $\boldsymbol{\Omega}_+ = \begin{bmatrix}\boldsymbol{\Omega} & \boldsymbol{\omega}\end{bmatrix}$, where $\boldsymbol{\omega} \in \mathbb{F}^n$ is a standard normal vector independent from $\boldsymbol{\Omega}$. Define $\boldsymbol{P}_+ \in \mathbb{S}_n$ to be the orthoprojector onto $\operatorname{range}(\boldsymbol{X}^{1/2}\boldsymbol{\Omega}_+)$. It is clear that $\operatorname{range}(\boldsymbol{P}) \subseteq \operatorname{range}(\boldsymbol{P}_+)$, and so

$$\boldsymbol{X}^{1/2}\boldsymbol{P}\boldsymbol{X}^{1/2} \preccurlyeq \boldsymbol{X}^{1/2}\boldsymbol{P}_+\boldsymbol{X}^{1/2}.$$

As a consequence,

$$\operatorname{tr}([\![\boldsymbol{X}^{1/2}\boldsymbol{P}\boldsymbol{X}^{1/2}]\!]_r) \leq \operatorname{tr}([\![\boldsymbol{X}^{1/2}\boldsymbol{P}_+\boldsymbol{X}^{1/2}]\!]_r).$$

Equivalently,

$$\|\boldsymbol{X} - [\![\boldsymbol{X}^{1/2}\boldsymbol{P}\boldsymbol{X}^{1/2}]\!]_r\|_* \geq \|\boldsymbol{X} - [\![\boldsymbol{X}^{1/2}\boldsymbol{P}_+\boldsymbol{X}^{1/2}]\!]_r\|_*.$$

Take the expectation with respect to $\boldsymbol{\Omega}_+$. The left-hand side is the average error in the r-truncated Nyström approximation with a standard normal sketch of size R. The right-hand side is the same thing, except the sketch has size $R+1$. This is the required result.

Appendix C. SketchyCGAL: Additional results.

This section contains some additional material about the SketchyCGAL algorithm.

C.1. Solution quality.

We can develop estimates for the quality of the SketchyCGAL solution by adapted the approach that we used for CGAL.

To do so, we need to track the primal objective value at the current iterate:

$$p_t = \langle \boldsymbol{C}, \boldsymbol{X}_t \rangle.$$

At each iteration, we can easily update this estimate using the computed approximate eigenvector $\boldsymbol{v}_t$:

$$p_{t+1} = (1-\eta_t)p_t + \eta_t \alpha \langle \boldsymbol{v}_t, \boldsymbol{C}\boldsymbol{v}_t \rangle.$$

This update rule is applied with the help of the primitive (2.4)❶.

When we wish to estimate the error, say in iteration t, we solve the eigenvalue subproblem to very high accuracy:

$$\xi_t = \boldsymbol{v}_t^* \boldsymbol{D}_t \boldsymbol{v}_t = \min_{\|\boldsymbol{v}\|=1} \boldsymbol{v}^* \boldsymbol{D}_t \boldsymbol{v}.$$

Then, we can compute the surrogate duality gap:

$$g_t(\boldsymbol{X}_t) = p_t + \langle \boldsymbol{y}_t + \beta_t(\boldsymbol{z}_t - \boldsymbol{b}),\, \boldsymbol{z}_t \rangle - \xi_t.$$

This expression follows directly from the formula (A.32) using the loop invariant that $\boldsymbol{z}_t = \mathcal{A}\boldsymbol{X}_t$. We arrive at a computable error estimate:

$$p_t - p_\star \leq g_t(\boldsymbol{X}_t) - \langle \boldsymbol{y}_t,\, \boldsymbol{z}_t - \boldsymbol{b} \rangle.$$

This bound follows directly from (A.33).

C.2. Convergence theory.

In this section, we establish two simple results on the convergence properties of the SketchyCGAL algorithm.

Theorem C.1 (SketchyCGAL: Convergence I). *Let $\Psi_\star$ be the solution set of the model problem (2.2). For each $r < R$, the iterates $\widehat{\boldsymbol{X}}_t$ computed by SketchyCGAL (subsections 6.2 and 6.3) satisfy*

$$\limsup_{t\to\infty} \mathbb{E}_{\boldsymbol{\Omega}} \operatorname{dist}_*(\widehat{\boldsymbol{X}}_t, \Psi_\star) \leq \left(1 + \frac{r}{R-r-1}\right) \cdot \max_{\boldsymbol{X}\in\Psi_\star} \|\boldsymbol{X} - [\![\boldsymbol{X}]\!]_r\|_*.$$

The same bound holds for the truncated approximations $[\![\widehat{\boldsymbol{X}}_t]\!]_r$. Here, dist_* is the nuclear-norm distance between a matrix and a set of matrices.

Proof. The implicit iterates $\boldsymbol{X}_t$ satisfy the conclusions of Fact 3.1, so they converge toward the compact set $\Psi_\star$. Therefore, we can choose a sequence $\{\boldsymbol{X}_{t\star}\} \subset \Psi_\star$ with the property that $\|\boldsymbol{X}_t - \boldsymbol{X}_{t\star}\|_* \to 0$. By the triangle inequality and (6.9),

$$\begin{aligned}\mathbb{E}_{\boldsymbol{\Omega}} \operatorname{dist}_*(\widehat{\boldsymbol{X}}_t, \Psi_\star) &\le \mathbb{E}_{\boldsymbol{\Omega}} \|\widehat{\boldsymbol{X}}_t - \boldsymbol{X}_t\|_* + \operatorname{dist}_*(\boldsymbol{X}_t, \Psi_\star) \\ &\le \left(1 + \frac{r}{R - r - 1}\right) \cdot \|\boldsymbol{X}_t - [\![\boldsymbol{X}_t]\!]_r\|_* + \|\boldsymbol{X}_t - \boldsymbol{X}_{t\star}\|_*.\end{aligned}$$

The rank-r approximation error in Schatten 1-norm is 1-Lipschitz with respect to the Schatten 1-norm (cf. [102, Sec. SM2.2]), so

$$\|\boldsymbol{X}_t - [\![\boldsymbol{X}_t]\!]_r\|_* \le \|\boldsymbol{X}_{t\star} - [\![\boldsymbol{X}_{t\star}]\!]_r\|_* + \|\boldsymbol{X}_t - \boldsymbol{X}_{t\star}\|_* \le \max_{\boldsymbol{X} \in \Psi_\star} \|\boldsymbol{X} - [\![\boldsymbol{X}]\!]_r\|_* + \|\boldsymbol{X}_t - \boldsymbol{X}_{t\star}\|_*.$$

Combine the last two displays, and extract the superior limit. ∎

If the implicit iterates generated by SketchyCGAL happen to converge to a limit, we have a more precise result.

Theorem C.2 (SketchyCGAL: Convergence II). *Assume the implicit iterates $\boldsymbol{X}_t$ induced by SketchyCGAL (subsections 6.2 and 6.3) converge to a matrix $\boldsymbol{X}_{\mathrm{cgal}}$. For each $r < R$, the computed iterates $\widehat{\boldsymbol{X}}_t$ satisfy*

$$\begin{aligned}\limsup_{t\to\infty} \mathbb{E}_{\boldsymbol{\Omega}} \|\mathcal{A}\widehat{\boldsymbol{X}}_t - \boldsymbol{b}\| &\le \left(1 + \frac{r}{R - r - 1}\right) \cdot \|\mathcal{A}\| \cdot \|\boldsymbol{X}_{\mathrm{cgal}} - [\![\boldsymbol{X}_{\mathrm{cgal}}]\!]_r\|_*; \\ \limsup_{t\to\infty} \mathbb{E}_{\boldsymbol{\Omega}} |\langle \boldsymbol{C}, \widehat{\boldsymbol{X}}_t\rangle - \langle \boldsymbol{C}, \boldsymbol{X}_\star\rangle| &\le \left(1 + \frac{r}{R - r - 1}\right) \cdot \|\boldsymbol{C}\| \cdot \|\boldsymbol{X}_{\mathrm{cgal}} - [\![\boldsymbol{X}_{\mathrm{cgal}}]\!]_r\|_*.\end{aligned}$$

The same bound holds for the truncated approximations $[\![\widehat{\boldsymbol{X}}_t]\!]_r$. If $\operatorname{rank}(\boldsymbol{X}_{\mathrm{cgal}}) \le R$, then the computed iterates $\widehat{\boldsymbol{X}}_t$ converge to the solution set of (2.2).

Proof. The implicit iterates $\boldsymbol{X}_t$ satisfy the conclusions of Fact 3.1, so the limit $\boldsymbol{X}_{\mathrm{cgal}}$ solves (2.2). Using the triangle inequality, the operator norm bound, and (6.9), we obtain nonasymptotic error bounds

$$\begin{aligned}\mathbb{E}_{\boldsymbol{\Omega}} \|\mathcal{A}\widehat{\boldsymbol{X}}_t - \boldsymbol{b}\| &\le \frac{\mathit{Const}}{\sqrt{t}} + \left(1 + \frac{r}{R - r - 1}\right) \cdot \|\mathcal{A}\| \cdot \|\boldsymbol{X}_t - [\![\boldsymbol{X}_t]\!]_r\|_*; \\ \mathbb{E}_{\boldsymbol{\Omega}} |\langle \boldsymbol{C}, \widehat{\boldsymbol{X}}_t\rangle - \langle \boldsymbol{C}, \boldsymbol{X}_\star\rangle| &\le \frac{\mathit{Const}}{\sqrt{t}} + \left(1 + \frac{r}{R - r - 1}\right) \cdot \|\boldsymbol{C}\| \cdot \|\boldsymbol{X}_t - [\![\boldsymbol{X}_t]\!]_r\|_*.\end{aligned}$$

Extract the limit as $t \to \infty$. The last conclusion follows from the facts outlined in Appendix B.5. ∎

Appendix D. Beyond the model problem.

The CGAL algorithm [110] applies to a more general problem template than (2.2). Likewise, the SketchyCGAL algorithm can solve a wider class of problems in a scalable fashion. This section outlines some of the opportunities.

D.1. A more general template. Consider the optimization problem

$$\text{(D.1)} \qquad \text{minimize} \quad \langle \boldsymbol{C}, \boldsymbol{X} \rangle \quad \text{subject to} \quad \mathcal{A}\boldsymbol{X} \in \mathsf{K} \quad \text{and} \quad \boldsymbol{X} \in \mathsf{X}, \quad \boldsymbol{X} \text{ is psd.}$$

In this expression, $\mathsf{K} \subset \mathbb{R}^d$ is a closed, convex set and $\mathsf{X} \subset \mathbb{S}_n(\mathbb{F})$ is a compact, convex set of matrices. The rest of this section describes some problems that fall within the compass of (D.1), as well as new computational challenges that appear. Algorithm D.1 contains pseudocode for a version of SketchyCGAL tailored to (D.1).

Remark D.1 (Other matrix optimization problems). We can also extend SketchyCGAL to optimization problems involving matrices that are symmetric (but not psd) or that are rectangular. For example, matrix completion via nuclear-norm minimization [96] falls in this framework. In this case, we need to replace the Nyström sketch with a more general technique, such as [101, 102]. Further extensions are also possible; see [110]. We omit these developments.

D.2. The convex constraint set. To handle the convex constraint X that appears in (D.1), we must develop a subroutine for the linear minimization problem

$$\text{(D.2)} \qquad \underset{\boldsymbol{H} \in \mathbb{S}_n}{\text{minimize}} \quad \langle \boldsymbol{D}_t, \boldsymbol{H} \rangle \quad \text{subject to} \quad \boldsymbol{H} \in \mathsf{X}, \quad \boldsymbol{H} \text{ is psd.}$$

To implement SketchyCGAL efficiently, we need the problem (D.2) to admit a structured (e.g., low-rank) approximate solution. Here are some situations where this is possible.

1. **Trace-bounded psd matrices.** $\mathsf{X} := \{\boldsymbol{X} \in \mathbb{S}_n : \operatorname{tr} \boldsymbol{X} \leq \alpha \text{ and } \boldsymbol{X} \text{ is psd}\}$. For solving a standard-form SDP, this constraint is more natural than $\mathsf{X} = \alpha\boldsymbol{\Delta}_n$. Given an (approximate) minimum eigenpair $(\xi_t, \boldsymbol{v}_t)$ of $\boldsymbol{D}_t$, the solution of (D.2) is
$$\boldsymbol{H}_t = \begin{cases} \alpha \boldsymbol{v}_t \boldsymbol{v}_t^*, & \xi_t < 0, \\ \boldsymbol{0}, & \xi_t \geq 0. \end{cases}$$
As before, we can solve the eigenvector problem with Algorithms 4.1 and 4.2.
2. **Relaxed orthoprojectors.** $\mathsf{X} := \{\boldsymbol{X} \in \mathbb{S}_n : \operatorname{tr} \boldsymbol{X} = \alpha \text{ and } \boldsymbol{0} \preccurlyeq \boldsymbol{X} \preccurlyeq \boldsymbol{I}\}$. This is the best convex relaxation of the set of orthogonal projectors with rank α; see [85]. When α is small, we can provably solve the linear minimization with randomized subspace iteration [49] or randomized block Lanczos methods [48, Sec. 10.3.6].

D.3. Convex inclusions. To handle the inclusion in K that appears in (D.1), we need an efficient algorithm to perform the Euclidean projection onto K. That is,

$$\operatorname{proj}_{\mathsf{K}}(\boldsymbol{w}) := \arg\min\{\|\boldsymbol{w} - \boldsymbol{u}\| : \boldsymbol{u} \in \mathsf{K}\} \quad \text{for } \boldsymbol{w} \in \mathbb{R}^d.$$

Here are some important examples:

1. **Inequality constraints.** $\mathsf{K} := \{\boldsymbol{u} \in \mathbb{R}^d : \boldsymbol{u} \leq \boldsymbol{b}\}$. In this case, the projection takes the form $\operatorname{proj}_{\mathsf{K}}(\boldsymbol{w}) = (\boldsymbol{w} - \boldsymbol{b})_-$, where $(\cdot)_-$ reports the negative part of a vector.
2. **Norm constraints.** $\mathsf{K} := \{\boldsymbol{u} \in \mathbb{R}^d : \|\!|\boldsymbol{u} - \boldsymbol{b}|\!\| \leq \delta\}$, where $\|\!|\cdot|\!\|$ is a norm. The projector can be computed easily for many norms, including the ℓ_p norm for $p \in \{1, 2, \infty\}$.

D.4. The CGAL iteration for the general template. To extend the description of the CGAL iteration in Appendix A.3 for the general template (D.1), we consider the following augmented Lagrangian formulation with the slack variable $\boldsymbol{w} \in \mathsf{K}$ instead of (A.6):

$$L_t(\boldsymbol{X}; \boldsymbol{y}) := \langle \boldsymbol{C}, \boldsymbol{X} \rangle + \min_{\boldsymbol{w} \in \mathsf{K}} \left\{ \langle \boldsymbol{y}, \mathcal{A}\boldsymbol{X} - \boldsymbol{w} \rangle + \frac{1}{2}\beta_t \|\mathcal{A}\boldsymbol{X} - \boldsymbol{w}\|^2 \right\}.$$

Accordingly, the partial derivative (A.8) becomes

$$\boldsymbol{D}_t := \partial_{\boldsymbol{X}} L_t(\boldsymbol{X}_t; \boldsymbol{y}_t) = \boldsymbol{C} + \mathcal{A}^*\big(\boldsymbol{y}_t + \beta_t(\mathcal{A}\boldsymbol{X}_t - \boldsymbol{w}_t)\big) \quad \text{where} \quad \boldsymbol{w}_t := \mathrm{proj}_{\mathsf{K}}(\mathcal{A}\boldsymbol{X}_t + \beta_t^{-1}\boldsymbol{y}_t).$$

We replace the linear minimization subroutine (A.9) with (D.2). We can still use an inexact variant of (D.2) with additive error. We also modify the dual update scheme by modifying (A.11) as

$$\boldsymbol{y}_{t+1} = \boldsymbol{y}_t + \gamma_t(\mathcal{A}\boldsymbol{X}_{t+1} - \overline{\boldsymbol{w}}_t) \quad \text{where} \quad \overline{\boldsymbol{w}}_t := \mathrm{proj}_{\mathsf{K}}(\mathcal{A}\boldsymbol{X}_{t+1} + \beta_{t+1}^{-1}\boldsymbol{y}_t).$$

Finally, we replace the dual step size parameter selection rule (A.12) with

$$\gamma_t \|\mathcal{A}\boldsymbol{X}_{t+1} - \overline{\boldsymbol{w}}_t\|^2 \le \beta_t \eta_t^2 \alpha^2 \|\mathcal{A}\|^2. \tag{D.3}$$

The bounded travel condition (A.13) remains the same.

To obtain the extension of SketchyCGAL to the general template, we simply pursue the same program outlined in section 6 to augment CGAL with sketching.

Appendix E. Details of phase retrieval experiments. This section presents further details about the phase retrieval experiments presented in section 7.

E.1. Synthetic phase retrieval data. This section provides additional details on the construction of synthetic datasets for the abstract phase retrieval SDP.

For each $n \in \{10^2, 10^3, \ldots, 10^6\}$, we generate 20 independent datasets as follows. First, draw $\boldsymbol{\chi}_\natural \in \mathbb{C}^n$ from the complex standard normal distribution. We acquire $d = 12n$ phaseless measurements (7.2) using the coded diffraction pattern model [31].

To do so, we randomly draw 12 independent modulating waveforms $\boldsymbol{\psi}_j$ for $j = 1, 2, \ldots, 12$. Each entry of $\boldsymbol{\psi}_j$ is drawn as the product of two independent random variables, one chosen uniformly from $\{1, i, -1, -i\}$, and the other from $\{\sqrt{2}/2, \sqrt{3}\}$ with probabilities 0.8 and 0.2 respectively. Then, we modulate $\boldsymbol{\chi}_\natural$ with these waveforms and take its Fourier transform. Each $\boldsymbol{a}_i$ corresponds to computing a single entry of this Fourier transform:

$$\boldsymbol{a}_{(j-1)n+\ell} = \boldsymbol{W}_n(\ell, :)\ \mathrm{diag}^*(\boldsymbol{\psi}_j), \qquad 1 \le j \le 12 \quad \text{and} \quad 1 \le \ell \le n,$$

where $\boldsymbol{W}_n(\ell, :)$ is the ℓth row of the $n \times n$ discrete Fourier transform matrix. We use the fast Fourier transform to implement the measurement operator.

E.2. Fourier ptychography. We study a more realistic measurement setup, Fourier ptychography (FP), for the phase retrieval problem. In this setup, $\boldsymbol{\chi}_\natural \in \mathbb{C}^n$ corresponds to an unknown high resolution image (vectorized) from a microscopic sample in the Fourier domain.

Algorithm D.1 SketchyCGAL for the general template (D.1)

Input: Problem data for (D.1) implemented via the primitives (2.4), sketch size R, number T of iterations

Output: Rank-R approximate solution to (D.1) in factored form $\widehat{\boldsymbol{X}}_T = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^*$ where $\boldsymbol{U} \in \mathbb{R}^{n\times R}$ has orthonormal columns and $\boldsymbol{\Lambda} \in \mathbb{R}^{R\times R}$ is nonnegative diagonal, and the Schatten 1-norm approximation errors $\mathrm{err}(\llbracket\widehat{\boldsymbol{X}}\rrbracket_r)$ for $1 \le r \le R$, as defined in (B.4)

Recommendation: To achieve (2.3), set R as large as possible, and set $T \approx \varepsilon^{-1}$

function SketchyCGAL(R; T)
  Scale problem data (subsection 7.1.2) ▷ **[opt]** Recommended!
  $\beta_0 \leftarrow 1$ and $K \leftarrow +\infty$ ▷ Default parameters
  NystromSketch.Init(n, R)
  $\boldsymbol{z} \leftarrow \boldsymbol{0}_d$ and $\boldsymbol{y} \leftarrow \boldsymbol{0}_d$
  for $t \leftarrow 1, 2, 3, \ldots, T$ **do**
    $\beta \leftarrow \beta_0\sqrt{t+1}$ and $\eta \leftarrow 2/(t+1)$
    $\boldsymbol{w} \leftarrow \mathrm{proj}_{\mathsf{K}}(\boldsymbol{z} + \beta^{-1}\boldsymbol{y})$
    $\boldsymbol{D} \leftarrow \boldsymbol{C} + \mathcal{A}^*(\boldsymbol{y} + \beta(\boldsymbol{z} - \boldsymbol{w}))$ ▷ Represent via primitives (2.4)❶❷
    $\boldsymbol{H}$ is a (low-rank) matrix that solves (D.2)
    $\boldsymbol{z} \leftarrow (1-\eta)\,\boldsymbol{z} + \eta\,\mathcal{A}(\boldsymbol{H})$ ▷ Use primitive (2.4)❸
    $\beta_+ \leftarrow \beta_0\sqrt{t+2}$
    $\boldsymbol{w} \leftarrow \mathrm{proj}_{\mathsf{K}}(\boldsymbol{z} + \beta_+^{-1}\boldsymbol{y})$
    $\boldsymbol{y} \leftarrow \boldsymbol{y} + \gamma(\boldsymbol{z} - \boldsymbol{w})$ ▷ Step size γ satisfies (A.13) and (D.3)
    NystromSketch.RankOneUpdate($\sqrt{\alpha}\boldsymbol{v}$, η)
  $(\boldsymbol{U}, \boldsymbol{\Lambda}, \mathbf{err}) \leftarrow$ NystromSketch.Reconstruct

One cannot directly acquire a high resolution image from this sample because of the physical limitations of optical systems. Any measurement is subject to a filter caused by the lens aperture. We can represent this filter by a sparse matrix $\boldsymbol{\Phi} \in \mathbb{C}^{m\times n}$ with $m \le n$, each row of which has only one non-zero coefficient. Because of this filter, we can acquire only low-resolution images, through m-dimensional Fourier transform.

FP enlightens the sample from L different angles using a LED grid. This lets us to to obtain L different aperture matrices Φ_j, $j = 1, 2, \ldots, L$. Then, we acquire phaseless measurements from the sample using the following transmission matrices:

$$\boldsymbol{a}_{(j-1)m+\ell} = \boldsymbol{W}_m^*(\ell, :)\,\Phi_j, \qquad 1 \le j \le L \quad \text{and} \quad 1 \le \ell \le m.$$

Here, $\boldsymbol{W}_m^*(\ell, :)$ is the ℓth row of the conjugate transpose of discrete Fourier transform matrix.

The aim in FP is to reconstruct complex valued $\boldsymbol{\chi}_\natural$ from these phaseless measurements. Once we construct $\boldsymbol{\chi}_\natural$, we can generate a high resolution image by taking its inverse Fourier transform.

Appendix F. Additional numerical results.

This section contains quantitative results from the MaxCut and QAP experiments.

F.1. The MaxCut SDP. This section gives further information about the MaxCut experiments summarized in subsection 7.2. Table 3 presents numerical data from the MaxCut SDP experiment with GSET benchmark. We compare the methods in terms of the cut weight, objective value, primal and dual infeasibility, and the storage cost and computation time. The storage cost is approximated by monitoring the virtual memory size of the process, hence it includes the memory that is swapped out and it can go beyond 16 GB.

F.2. Failure of the Burer–Monteiro heuristic. This section presents empirical evidence that Burer–Monteiro factorization methods cannot support storage costs better than $\Omega(n\sqrt{d})$. Our approach is based on the paper of Waldspurger & Waters [105], which proves that the BM heuristic (8.2) can produce incorrect results unless $R = \Omega(n\sqrt{d})$.

Waldspurger provided us code that generates a random symmetric $\boldsymbol{C} \in \mathbb{S}_n(\mathbb{R})$. The MaxCut SDP (1.3) with objective $\boldsymbol{C}$ has a unique solution, and the solution has rank 1. If the factorization rank R satisfies $R(R+3) \leq 2n$, then the BM formulation (8.2) has second-order critical points that are not optimal points of the original SDP (1.3). As a consequence, the Burer–Monteiro approach is reliable only if the storage budget is $\Theta(n^{3/2})$. In contrast, for the same problem instances, our analysis (Theorem Theorem 6.3) shows that SketchyCGAL succeeds with factorization rank $R = 2$ and storage budget $\Theta(n)$.

We will demonstrate numerically that Waldspurger & Waters [105] have identified a serious obstruction to using the Burer–Monteiro approach. Moreover, we will see that SketchyCGAL resolves the issue. See the code supplement for scripts to reproduce these experiments.

We use the Manopt software [22] to solve the Burer–Monteiro formulation of the MaxCut SDP. For $n = 100$, we drew 10 random matrices $\boldsymbol{C}_1, \ldots, \boldsymbol{C}_{10}$ using Waldspurger's code. For each instance, we sweep the factorization rank $R = 2, 3, 4, \ldots, 13$. (For $R \geq 13$, we anticipate that each second-order critical point of the Burer–Monteiro problem is a solution to the original SDP, owing to the analysis in [23].) In each experiment, we ran Manopt with 100 random initializations, and we counted the number of times the algorithm failed. We declared failure if Manopt converged to a second-order stationary point whose objective value is 10^{-3} larger than the true optimal value. See Table 2 for the statistics.

In contrast, SketchyCGAL can solve all of these instances, even when the sketch size $R = 2$. For these problems, we use the default parameter choices for SketchyCGAL, but we do not pre-scale the data or perform tuning. Figure F.1 compares the convergence trajectory of SketchyCGAL and Manopt for one problem instance. The difference is evident.

F.3. The quadratic assignment problem. This section gives further information about the QAP experiments summarized in subsection 7.5. Tables 5 and 7 display the performance of SketchyCGAL, by presenting the upper bound after rounding (subsection 7.5.3), the objective value $\langle \boldsymbol{B} \otimes \boldsymbol{A}, \boldsymbol{X} \rangle$, the feasibility gap $\mathrm{dist}_{\mathsf{K}}(\mathcal{A}\boldsymbol{X})$, the total number of iterations, the memory usage, and the computation time.

Tables 4 and 6 compare the relative gap (7.7) obtained by SketchyCGAL with the values for the CSDP method [26] with clique size $k = \{2, 3, 4\}$ and the PATH method [116] reported in [26, Tab. 6]. These results appeared also in Figure 7.5.

Acknowledgments. We would like to thank Irène Waldspurger for providing code that generates instances of the MaxCut SDP that are difficult for the Burer–Monteiro heuristic.

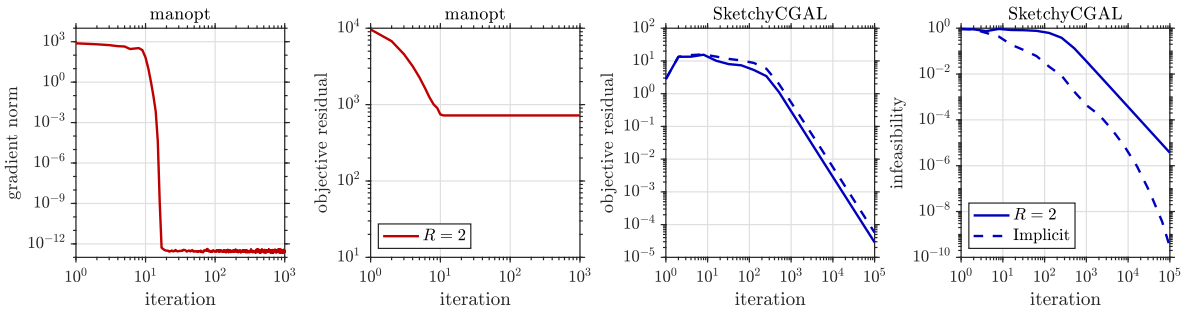


Figure F.1. MaxCut SDP: Failure of the Burer–Monteiro heuristic. *We apply Manopt and SketchyCGAL for solving the MaxCut SDP with the dataset $\boldsymbol{C}_1$. The subplots show the gradient norm and suboptimality for Manopt [left] and the suboptimality and infeasibility for SketchyCGAL [right]. Manopt with $R = 2$ converges to a spurious solution, whereas SketchyCGAL successfully computes a rank-1 approximation of the global optimum. The dashed line describes the convergence of the SketchyCGAL implicit iterates. For details, see Appendix F.2.*

Table 2

We run Manopt for solving hard instances of the MaxCut SDP. We consider 10 datasets $\boldsymbol{C}_1, \ldots, \boldsymbol{C}_{10}$. For each dataset, we run Manopt with 100 random initializations and report the number of failures. We declare failure when Manopt converges to second-order critical point that is not a global optimum. For details, see Appendix F.2.

Dataset / R	$R = 2$	3	4	5	6	7	8	9	10	11	12	13
$\boldsymbol{C}_1$	82	69	63	53	35	32	24	12	11	1	4	0
$\boldsymbol{C}_2$	77	56	56	36	19	17	12	2	0	0	0	0
$\boldsymbol{C}_3$	89	65	54	47	44	46	23	11	5	0	3	0
$\boldsymbol{C}_4$	84	69	50	40	27	23	18	17	1	0	9	0
$\boldsymbol{C}_5$	85	68	52	51	43	30	31	20	14	3	4	0
$\boldsymbol{C}_6$	81	68	53	41	23	22	10	10	2	0	1	0
$\boldsymbol{C}_7$	83	76	60	39	19	19	19	3	0	0	1	0
$\boldsymbol{C}_8$	81	73	44	34	41	25	8	12	5	4	10	0
$\boldsymbol{C}_9$	84	64	46	35	25	17	1	10	0	2	4	0
$\boldsymbol{C}_{10}$	83	71	54	50	31	25	24	16	13	0	8	0

Table 3: Numerical outcomes from the MaxCut SDP experiment with GSET Benchmark.

	SketchyCGAL	MoSeK	SDPT3	SDPNAL+	Sedumi
Data Name Size (n)	cut weight $\langle \boldsymbol{C}, \boldsymbol{X} \rangle$ $\|\mathcal{A}\boldsymbol{X} - \boldsymbol{b}\| / \max\{\|\boldsymbol{b}\|, 1\}$ $\|\min\{\lambda_{\min}(\boldsymbol{X}), 0\}\|$ storage (MB) cpu time (hh:mm:ss)				
G1 800	11382 -1.2072e+04 2.7955e-03 0 7 00:00:00:03	11414 -1.2083e+04 3.0760e-08 0 225 00:00:00:15	11414 -1.2083e+04 9.3301e-13 0 180 00:00:00:15	11414 -1.2083e+04 8.6224e-15 0 258 00:00:01:35	11414 -1.2083e+04 3.6902e-09 0 354 00:00:00:43
G2 800	11361 -1.2082e+04 2.8124e-03 0 7 00:00:00:03	11329 -1.2089e+04 3.4843e-08 0 225 00:00:00:16	11329 -1.2089e+04 9.5745e-13 0 180 00:00:00:12	11339 -1.2089e+04 3.3489e-15 0 275 00:00:01:43	11329 -1.2089e+04 2.9720e-09 0 352 00:00:00:41
G3 800	11388 -1.2076e+04 2.5103e-03 0 7 00:00:00:03	11392 -1.2084e+04 5.9077e-08 0 225 00:00:00:19	11392 -1.2084e+04 1.7491e-11 0 180 00:00:00:13	11392 -1.2084e+04 3.4555e-07 0 219 00:00:01:15	11392 -1.2084e+04 3.9665e-09 0 356 00:00:00:51
G4 800	11387 -1.2103e+04 2.6474e-03 0 7 00:00:00:05	11366 -1.2111e+04 4.2044e-08 0 225 00:00:00:14	11366 -1.2111e+04 1.8551e-13 0 180 00:00:00:11	11368 -1.2111e+04 3.3027e-07 0 277 00:00:01:16	11366 -1.2111e+04 3.3830e-09 0 354 00:00:00:52
G5 800	11356 -1.2092e+04 2.6573e-03 0 7 00:00:00:03	11432 -1.2100e+04 4.2128e-08 0 225 00:00:00:14	11432 -1.2100e+04 3.1188e-13 0 180 00:00:00:12	11432 -1.2100e+04 6.3811e-09 0 270 00:00:01:50	11432 -1.2100e+04 3.2630e-09 0 352 00:00:00:45
G6 800	1899 -2.6534e+03 2.7480e-03 0 7 00:00:00:06	1950 -2.6562e+03 2.8773e-08 0 225 00:00:00:15	1950 -2.6562e+03 3.2757e-12 0 180 00:00:00:16	1950 -2.6562e+03 1.0055e-07 0 273 00:00:01:17	1950 -2.6562e+03 1.7342e-09 0 354 00:00:00:43
G7 800	1779 -2.4873e+03 2.7858e-03 0 7 00:00:00:04	1749 -2.4893e+03 2.3310e-08 0 225 00:00:00:22	1749 -2.4893e+03 2.5344e-12 0 180 00:00:00:16	1749 -2.4893e+03 1.2441e-07 0 268 00:00:01:14	1749 -2.4893e+03 1.6999e-09 0 352 00:00:01:06

Continue on the next page

Table 3: MaxCut Benchmark with GSet (cont.).

	SketchyCGAL	MoSeK	SDPT3	SDPNAL+	Sedumi
G8 800	1784	1765	1765	1765	1765
	-2.5056e+03	-2.5069e+03	-2.5069e+03	-2.5069e+03	-2.5069e+03
	2.7816e-03	4.2417e-08	3.4138e-13	8.5771e-07	1.7960e-09
	0	0	0	0	0
	7	225	180	277	289
	00:00:00:04	00:00:00:15	00:00:00:15	00:00:01:25	00:00:00:45
G9 800	1776	1800	1800	1800	1800
	-2.5262e+03	-2.5287e+03	-2.5287e+03	-2.5287e+03	-2.5287e+03
	2.7310e-03	2.9469e-08	8.2962e-12	2.4830e-07	3.9707e-10
	0	0	0	0	0
	7	225	180	277	352
	00:00:00:06	00:00:00:14	00:00:00:12	00:00:01:08	00:00:00:46
G10 800	1790	1766	1766	1766	1766
	-2.4840e+03	-2.4851e+03	-2.4851e+03	-2.4851e+03	-2.4851e+03
	2.7810e-03	3.7738e-08	5.7206e-13	7.4784e-07	2.9698e-10
	0	0	0	0	0
	7	225	180	277	352
	00:00:00:04	00:00:00:15	00:00:00:12	00:00:01:21	00:00:01:04
G11 800	524	512	512	522	512
	-6.2814e+02	-6.2916e+02	-6.2916e+02	-6.2914e+02	-6.2916e+02
	2.6719e-03	1.7634e-13	1.4007e-12	1.0682e-10	1.8847e-09
	0	0	0	0	0
	7	225	186	277	352
	00:00:00:04	00:00:00:19	00:00:00:09	00:00:01:52	00:00:00:39
G12 800	524	512	512	514	512
	-6.2279e+02	-6.2387e+02	-6.2387e+02	-6.2387e+02	-6.2387e+02
	2.6153e-03	1.8468e-13	5.3073e-13	1.8087e-11	1.1260e-09
	0	0	0	0	0
	7	225	186	278	352
	00:00:00:03	00:00:00:14	00:00:00:07	00:00:02:20	00:00:00:46
G13 800	554	534	534	534	534
	-6.4594e+02	-6.4714e+02	-6.4714e+02	-6.4713e+02	-6.4714e+02
	2.6652e-03	1.6420e-13	8.3034e-12	3.4895e-15	7.7471e-10
	0	0	0	0	0
	7	225	186	278	354
	00:00:00:03	00:00:00:14	00:00:00:08	00:00:01:47	00:00:00:41
G14 800	2953	2967	2967	2963	2967
	-3.1821e+03	-3.1916e+03	-3.1916e+03	-3.1916e+03	-3.1916e+03
	2.4329e-03	3.4503e-08	4.1785e-13	7.3214e-09	1.1476e-09
	0	0	0	0	0
	7	225	180	284	352
	00:00:00:05	00:00:00:16	00:00:00:09	00:00:01:47	00:00:00:49
G15 800	2948	2971	2971	2971	2971
	-3.1611e+03	-3.1716e+03	-3.1716e+03	-3.1716e+03	-3.1716e+03
	2.1250e-03	5.9551e-08	9.9206e-14	3.8762e-07	1.6534e-09
	0	0	0	0	0
	7	225	186	278	352
	00:00:00:04	00:00:00:15	00:00:00:13	00:00:02:23	00:00:00:45

Continue on the next page

Table 3: MaxCut Benchmark with GSet (cont.).

	SketchyCGAL	MoSeK	SDPT3	SDPNAL+	Sedumi
	2967	2958	2958	2956	2958
	-3.1658e+03	-3.1750e+03	-3.1750e+03	-3.1750e+03	-3.1750e+03
G16	2.3354e-03	8.1927e-08	4.7805e-13	9.3060e-15	3.1805e-10
800	0	0	0	0	0
	7	225	180	279	352
	00:00:00:04	00:00:00:20	00:00:00:10	00:00:02:00	00:00:00:46
	2938	2957	2957	2960	2957
	-3.1621e+03	-3.1713e+03	-3.1713e+03	-3.1713e+03	-3.1713e+03
G17	2.4028e-03	1.3790e-08	1.2021e-12	9.2606e-07	7.2377e-10
800	0	0	0	0	0
	7	225	186	280	352
	00:00:00:04	00:00:00:17	00:00:00:10	00:00:02:17	00:00:00:49
	898	893	893	893	893
	-1.1643e+03	-1.1660e+03	-1.1660e+03	-1.1660e+03	-1.1660e+03
G18	2.8264e-03	1.0695e-08	1.2377e-11	9.8235e-07	1.2709e-09
800	0	0	0	0	0
	7	225	186	278	352
	00:00:00:06	00:00:00:17	00:00:00:09	00:00:01:37	00:00:01:06
	792	797	797	797	797
	-1.0790e+03	-1.0820e+03	-1.0820e+03	-1.0820e+03	-1.0820e+03
G19	2.7808e-03	1.4813e-08	5.5509e-12	3.8521e-08	1.3036e-09
800	0	0	0	0	0
	7	225	186	273	352
	00:00:00:06	00:00:00:16	00:00:00:12	00:00:01:38	00:00:00:58
	846	837	837	837	837
	-1.1086e+03	-1.1114e+03	-1.1114e+03	-1.1114e+03	-1.1114e+03
G20	2.7607e-03	1.6650e-08	8.3457e-13	6.6975e-07	7.4903e-10
800	0	0	0	0	0
	7	225	186	285	354
	00:00:00:06	00:00:00:17	00:00:00:10	00:00:01:34	00:00:01:02
	811	841	841	841	841
	-1.1019e+03	-1.1043e+03	-1.1043e+03	-1.1043e+03	-1.1043e+03
G21	2.6359e-03	9.8852e-09	5.5307e-13	4.8156e-07	2.0021e-10
800	0	0	0	0	0
	7	225	186	268	357
	00:00:00:05	00:00:00:17	00:00:00:13	00:00:01:33	00:00:00:54
	12974	12956	12956	12955	12956
	-1.4125e+04	-1.4136e+04	-1.4136e+04	-1.4136e+04	-1.4136e+04
G22	4.3829e-03	1.0335e-08	7.0113e-13	1.9644e-07	2.7849e-09
2000	0	0	0	0	0
	8	1049	735	994	1498
	00:00:00:09	00:00:06:17	00:00:01:35	00:00:49:50	00:00:17:38
	12918	12987	12987	12996	12987
	-1.4133e+04	-1.4142e+04	-1.4142e+04	-1.4142e+04	-1.4142e+04
G23	4.3010e-03	1.6980e-08	1.6925e-14	1.6469e-09	8.3706e-10
2000	0	0	0	0	0
	8	1049	705	865	1401
	00:00:00:04	00:00:04:12	00:00:02:36	00:00:46:53	00:00:14:02

Continue on the next page

Table 3: MaxCut Benchmark with GSet (cont.).

	SketchyCGAL	MoSeK	SDPT3	SDPNAL+	Sedumi
G24 2000	12885	12979	12979	12978	12979
	-1.4135e+04	-1.4141e+04	-1.4141e+04	-1.4141e+04	-1.4141e+04
	4.4016e-03	1.0315e-08	1.1767e-12	2.8584e-09	7.7807e-10
	0	0	0	0	0
	8	1049	670	962	1498
	00:00:00:10	00:00:04:07	00:00:02:03	00:00:46:51	00:00:14:26
G25 2000	12913	12885	12885	12897	12885
	-1.4139e+04	-1.4144e+04	-1.4144e+04	-1.4144e+04	-1.4144e+04
	4.3776e-03	1.7277e-08	5.8895e-13	7.4190e-08	1.1674e-09
	0	0	0	0	0
	8	1049	705	993	1401
	00:00:00:17	00:00:06:31	00:00:01:31	00:00:46:09	00:00:16:08
G26 2000	12847	12866	12866	12866	12866
	-1.4122e+04	-1.4133e+04	-1.4133e+04	-1.4133e+04	-1.4133e+04
	4.3789e-03	1.0479e-08	5.0789e-14	4.6882e-08	1.0125e-09
	0	0	0	0	0
	8	1049	705	993	1466
	00:00:00:06	00:00:03:57	00:00:01:48	00:00:47:04	00:00:26:36
G27 2000	2863	2888	2888	2888	2888
	-4.1378e+03	-4.1417e+03	-4.1417e+03	-4.1417e+03	-4.1417e+03
	4.4638e-03	1.5899e-08	5.9893e-12	2.5844e-07	1.2743e-09
	0	0	0	0	0
	8	1049	768	1001	1466
	00:00:00:19	00:00:04:12	00:00:03:03	00:00:27:23	00:00:29:58
G28 2000	2856	2843	2843	2843	2843
	-4.0971e+03	-4.1008e+03	-4.1008e+03	-4.1008e+03	-4.1008e+03
	4.4429e-03	9.6101e-09	3.6450e-12	8.0676e-08	1.0742e-09
	0	0	0	0	0
	8	1049	734	1001	1432
	00:00:00:14	00:00:04:10	00:00:01:45	00:00:50:35	00:00:16:56
G29 2000	2999	2978	2978	2978	2978
	-4.2056e+03	-4.2089e+03	-4.2089e+03	-4.2089e+03	-4.2089e+03
	4.4590e-03	4.7152e-08	6.4617e-12	2.6237e-07	1.2095e-09
	0	0	0	0	0
	8	1049	705	973	1465
	00:00:00:13	00:00:04:32	00:00:01:49	00:00:46:07	00:00:26:32
G30 2000	2993	3007	3007	3007	3007
	-4.2107e+03	-4.2154e+03	-4.2154e+03	-4.2154e+03	-4.2154e+03
	4.3487e-03	2.8211e-08	1.1262e-11	3.6633e-07	1.3249e-09
	0	0	0	0	0
	8	1049	714	1033	1465
	00:00:00:07	00:00:07:03	00:00:02:54	00:00:29:38	00:00:27:07
G31 2000	2840	2876	2876	2876	2876
	-4.1139e+03	-4.1167e+03	-4.1167e+03	-4.1167e+03	-4.1167e+03
	4.4544e-03	1.3367e-08	4.0493e-11	9.9629e-07	1.4877e-09
	0	0	0	0	0
	8	1049	768	1040	1432
	00:00:00:09	00:00:04:01	00:00:02:27	00:00:52:58	00:00:20:18

Continue on the next page

Table 3: MaxCut Benchmark with GSet (cont.).

	SketchyCGAL	MoSeK	SDPT3	SDPNAL+	Sedumi
	1286	1268	1268	1288	1268
	-1.5645e+03	-1.5676e+03	-1.5676e+03	-1.5675e+03	-1.5676e+03
G32	4.4160e-03	2.1823e-13	3.5478e-11	2.9972e-14	8.8110e-10
2000	0	0	0	0	0
	8	1049	589	974	1466
	00:00:00:05	00:00:03:57	00:00:00:58	00:01:13:59	00:00:23:45
	1262	1276	1276	1282	1276
	-1.5410e+03	-1.5443e+03	-1.5443e+03	-1.5442e+03	-1.5443e+03
G33	4.4238e-03	1.8467e-13	2.1554e-11	5.0275e-15	2.4434e-09
2000	0	0	0	0	0
	8	1049	595	1036	1432
	00:00:00:05	00:00:03:36	00:00:01:02	00:01:20:22	00:00:12:44
	1266	1250	1254	1258	1256
	-1.5435e+03	-1.5467e+03	-1.5467e+03	-1.5466e+03	-1.5467e+03
G34	4.4154e-03	2.9281e-13	1.1258e-12	9.1706e-15	3.4482e-10
2000	0	0	0	0	0
	8	1049	595	1005	1466
	00:00:00:05	00:00:04:06	00:00:00:56	00:01:26:33	00:00:13:43
	7418	7411	7411	7411	7411
	-8.0057e+03	-8.0147e+03	-8.0147e+03	-8.0147e+03	-8.0147e+03
G35	4.4657e-03	2.3327e-08	1.3377e-11	5.6863e-09	6.6740e-10
2000	0	0	0	0	0
	8	1049	654	993	1494
	00:00:00:18	00:00:04:36	00:00:01:46	00:00:57:44	00:00:21:25
	7371	7399	7399	7396	7399
	-7.9941e+03	-8.0060e+03	-8.0060e+03	-8.0060e+03	-8.0060e+03
G36	4.4700e-03	3.5444e-08	2.6518e-14	2.2431e-09	4.4800e-10
2000	0	0	0	0	0
	8	1049	590	994	1498
	00:00:00:17	00:00:06:47	00:00:01:55	00:00:58:12	00:00:16:56
	7374	7443	7443	7437	7443
	-8.0007e+03	-8.0186e+03	-8.0186e+03	-8.0186e+03	-8.0186e+03
G37	4.4645e-03	3.7915e-08	8.6663e-12	7.5503e-09	2.0343e-09
2000	0	0	0	0	0
	8	1049	724	993	1401
	00:00:00:19	00:00:04:24	00:00:01:21	00:00:48:17	00:00:18:56
	7370	7402	7402	7400	7402
	-7.9999e+03	-8.0150e+03	-8.0150e+03	-8.0150e+03	-8.0150e+03
G38	4.4400e-03	2.7416e-08	4.4579e-13	8.8096e-09	2.0761e-10
2000	0	0	0	0	0
	8	1049	590	888	1431
	00:00:00:19	00:00:06:27	00:00:02:25	00:00:48:12	00:00:19:18
	2143	2176	2176	2171	2176
	-2.8700e+03	-2.8776e+03	-2.8776e+03	-2.8776e+03	-2.8776e+03
G39	4.4570e-03	1.0983e-08	1.7938e-12	1.7759e-09	5.1775e-10
2000	0	0	0	0	0
	8	1049	621	1067	1563
	00:00:00:15	00:00:04:37	00:00:01:49	00:00:47:07	00:00:32:23

Continue on the next page

Table 3: MaxCut Benchmark with GSet (cont.).

	SketchyCGAL	MoSeK	SDPT3	SDPNAL+	Sedumi
G40 2000	2071	2153	2153	2151	2153
	-2.8611e+03	-2.8648e+03	-2.8648e+03	-2.8648e+03	-2.8648e+03
	4.4469e-03	3.9814e-08	2.8335e-11	3.3046e-07	1.6695e-09
	0	0	0	0	0
	8	1049	723	1067	1532
	00:00:00:22	00:00:05:05	00:00:01:31	00:00:29:13	00:00:32:19
G41 2000	2087	2114	2114	2114	2114
	-2.8602e+03	-2.8652e+03	-2.8652e+03	-2.8652e+03	-2.8652e+03
	4.4599e-03	4.6842e-08	1.5034e-12	6.0544e-09	7.0174e-10
	0	0	0	0	0
	8	1049	641	1067	1498
	00:00:00:15	00:00:06:12	00:00:01:31	00:00:50:39	00:00:31:37
G42 2000	2186	2172	2172	2178	2172
	-2.9392e+03	-2.9463e+03	-2.9463e+03	-2.9463e+03	-2.9463e+03
	4.4565e-03	1.7392e-08	4.3777e-12	1.8720e-15	1.1033e-09
	0	0	0	0	0
	8	1049	671	936	1468
	00:00:00:12	00:00:06:28	00:00:01:28	00:00:25:19	00:00:21:00
G43 1000	6524	6518	6518	6518	6518
	-7.0268e+03	-7.0322e+03	-7.0322e+03	-7.0322e+03	-7.0322e+03
	3.0606e-03	4.4803e-08	1.1908e-13	6.6168e-07	1.3780e-09
	0	0	0	0	0
	7	298	246	347	483
	00:00:00:02	00:00:00:28	00:00:00:18	00:00:02:23	00:00:02:08
G44 1000	6491	6461	6461	6461	6461
	-7.0241e+03	-7.0279e+03	-7.0279e+03	-7.0279e+03	-7.0279e+03
	3.0788e-03	7.0255e-09	1.0885e-11	9.6232e-07	4.9394e-10
	0	0	0	0	0
	7	298	246	340	420
	00:00:00:02	00:00:00:28	00:00:00:22	00:00:02:26	00:00:01:28
G45 1000	6449	6447	6447	6447	6447
	-7.0208e+03	-7.0248e+03	-7.0248e+03	-7.0248e+03	-7.0248e+03
	3.0408e-03	1.0257e-08	9.2987e-12	9.7518e-07	6.2000e-10
	0	0	0	0	0
	7	298	246	348	420
	00:00:00:04	00:00:00:41	00:00:00:20	00:00:03:02	00:00:01:29
G46 1000	6458	6416	6416	6416	6416
	-7.0250e+03	-7.0299e+03	-7.0299e+03	-7.0299e+03	-7.0299e+03
	3.1575e-03	1.4069e-08	7.8291e-13	7.3592e-07	1.6385e-09
	0	0	0	0	0
	7	298	246	344	483
	00:00:00:04	00:00:00:32	00:00:00:18	00:00:02:23	00:00:01:45
G47 1000	6461	6452	6452	6454	6452
	-7.0324e+03	-7.0367e+03	-7.0367e+03	-7.0367e+03	-7.0367e+03
	3.0751e-03	4.6358e-09	5.9188e-12	6.8425e-15	1.1436e-09
	0	0	0	0	0
	7	298	246	348	482
	00:00:00:02	00:00:00:29	00:00:00:18	00:00:03:01	00:00:01:38

Continue on the next page

Table 3: MaxCut Benchmark with GSet (cont.).

	SketchyCGAL	MoSeK	SDPT3	SDPNAL+	Sedumi
G48	6000	6000	6000	6000	6000
3000	-5.9870e+03	-6.0000e+03	-6.0000e+03	-6.0812e+03	-6.0000e+03
	5.2439e-03	2.1863e-14	2.5288e-13	2.1707e-13	2.0368e-12
	0	0	0	0	0
	9	2086	959	1390	2621
	00:00:00:03	00:00:07:24	00:00:02:13	00:02:52:47	00:00:32:41
G49	6000	6000	6000	6000	6000
3000	-5.9871e+03	-6.0000e+03	-6.0000e+03	-6.0026e+03	-6.0000e+03
	5.4493e-03	1.4375e-13	3.3753e-14	3.9549e-13	4.4770e-10
	0	0	0	0	0
	9	2086	959	1424	2621
	00:00:00:02	00:00:07:44	00:00:01:49	00:01:53:02	00:00:43:07
G50	5868	5880	5880	5880	5880
3000	-5.9796e+03	-5.9882e+03	-5.9882e+03	-5.9915e+03	-5.9882e+03
	5.4017e-03	5.4964e-14	3.4115e-14	9.6276e-14	2.0697e-13
	0	0	0	0	0
	9	2152	959	1400	2621
	00:00:00:02	00:00:08:45	00:00:02:20	00:02:46:35	00:00:29:42
G51	3687	3738	3738	3737	3738
1000	-3.9918e+03	-4.0063e+03	-4.0063e+03	-4.0063e+03	-4.0063e+03
	2.2432e-03	6.5908e-08	4.7342e-13	9.3109e-07	1.4928e-09
	0	0	0	0	0
	7	298	246	348	483
	00:00:00:06	00:00:00:28	00:00:00:18	00:00:03:40	00:00:02:29
G52	3727	3711	3711	3708	3711
1000	-3.9956e+03	-4.0096e+03	-4.0096e+03	-4.0096e+03	-4.0096e+03
	2.3113e-03	1.1607e-08	1.1565e-12	1.0138e-06	2.1532e-09
	0	0	0	0	0
	7	298	246	347	418
	00:00:00:06	00:00:00:30	00:00:00:23	00:00:03:51	00:00:01:27
G53	3714	3712	3712	3712	3712
1000	-3.9972e+03	-4.0097e+03	-4.0097e+03	-4.0097e+03	-4.0097e+03
	1.9289e-03	6.2655e-08	6.4444e-14	1.0682e-14	4.4861e-10
	0	0	0	0	0
	7	298	246	355	483
	00:00:00:06	00:00:00:28	00:00:00:17	00:00:03:31	00:00:02:11
G54	3700	3716	3716	3718	3716
1000	-3.9934e+03	-4.0062e+03	-4.0062e+03	-4.0062e+03	-4.0062e+03
	1.9950e-03	1.4837e-08	1.4319e-12	1.5096e-14	1.5897e-09
	0	0	0	0	0
	7	298	311	347	482
	00:00:00:07	00:00:00:43	00:00:00:14	00:00:03:35	00:00:01:47
G55	9884	9866	9866	9865	9866
5000	-1.1036e+04	-1.1039e+04	-1.1039e+04	-1.1039e+04	-1.1039e+04
	4.0738e-03	6.5842e-09	1.0881e-12	2.1875e-07	1.9760e-09
	0	0	0	0	0
	11	6541	2646	3701	7448
	00:00:02:42	00:01:08:53	00:00:13:54	00:08:41:41	00:04:30:44

Continue on the next page

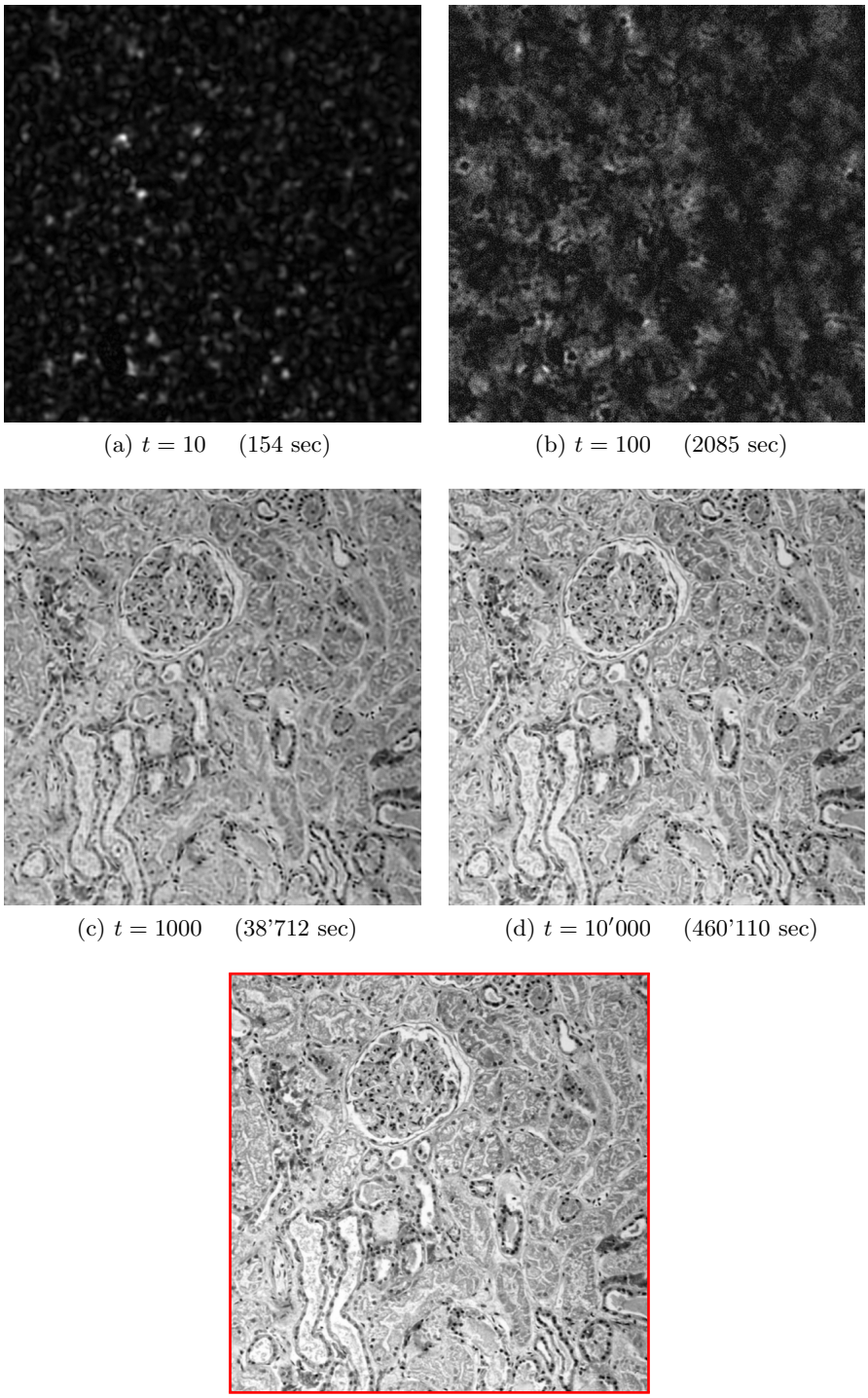
Table 3: MaxCut Benchmark with GSet (cont.).

	SketchyCGAL	MoSeK	SDPT3	SDPNAL+	Sedumi
	3584	3595	3595	3596	3595
	-4.7569e+03	-4.7600e+03	-4.7600e+03	-4.7600e+03	-4.7600e+03
G56	7.0679e-03	3.2421e-08	4.2710e-14	1.1317e-11	7.1008e-10
5000	0	0	0	0	0
	11	6541	2735	3700	6862
	00:00:01:49	00:01:05:47	00:00:19:58	00:13:19:39	00:04:12:45
	3150	3152	3156	3192	3158
	-3.8791e+03	-3.8855e+03	-3.8855e+03	-3.8853e+03	-3.8855e+03
G57	7.0600e-03	1.6506e-13	2.7378e-13	4.6605e-14	1.7408e-09
5000	0	0	0	0	0
	11	6541	2652	3700	6861
	00:00:00:16	00:00:55:50	00:00:13:39	00:20:51:45	00:03:49:29
	18470	18561	18561	18559	18561
	-2.0135e+04	-2.0136e+04	-2.0136e+04	-2.0136e+04	-2.0136e+04
G58	7.0648e-03	1.4230e-08	2.4310e-12	4.5090e-15	9.3470e-10
5000	0	0	0	0	0
	11	6541	2748	3724	6862
	00:00:02:52	00:01:17:25	00:00:18:26	00:14:15:52	00:04:40:20
	5321	5325	5325	5316	5325
	-7.2866e+03	-7.3123e+03	-7.3123e+03	-7.3123e+03	-7.3123e+03
G59	7.0688e-03	1.0705e-08	4.5632e-12	4.1272e-15	1.3622e-09
5000	0	0	0	0	0
	11	6541	2802	3734	6862
	00:00:01:46	00:01:14:29	00:00:19:02	00:18:10:33	00:09:19:12
	13617	13610	13610	13623	13610
	-1.5219e+04	-1.5222e+04	-1.5222e+04	-1.5222e+04	-1.5222e+04
G60	7.0468e-03	1.0901e-08	1.1320e-12	5.4916e-14	8.9220e-10
7000	0	0	0	0	0
	20	12681	5017	7074	13237
	00:00:04:04	00:02:27:21	00:00:53:31	01:10:39:19	00:11:31:27
	5208	5218	5218	5194	5218
	-6.8239e+03	-6.8281e+03	-6.8281e+03	-6.8281e+03	-6.8281e+03
G61	8.3648e-03	1.7210e-08	5.3273e-13	2.8846e-15	1.3178e-09
7000	0	0	0	0	0
	20	12681	5395	7137	13237
	00:00:02:09	00:02:37:09	00:00:41:19	01:08:33:11	00:17:32:16
	4406	4384	4384	4404	4384
	-5.4258e+03	-5.4309e+03	-5.4309e+03	-5.4306e+03	-5.4309e+03
G62	8.3568e-03	1.6462e-13	6.5021e-13	3.2675e-14	2.3616e-09
7000	0	0	0	0	0
	16	12681	5089	7071	13236
	00:00:00:56	00:02:04:21	00:00:35:16	01:13:44:46	00:09:23:00
	25890	25988	25988	26000	25988
	-2.8218e+04	-2.8244e+04	-2.8244e+04	-2.8244e+04	-2.8244e+04
G63	8.3635e-03	4.4392e-08	1.6210e-11	6.7982e-15	1.4974e-09
7000	0	0	0	0	0
	16	12681	5235	7077	13237
	00:00:06:04	00:02:48:34	00:00:52:03	01:17:16:18	00:12:58:10

Continue on the next page

Table 3: MaxCut Benchmark with GSet (cont.).

	SketchyCGAL	MoSeK	SDPT3	SDPNAL+	Sedumi
G64 7000	7645	7746	7746	7745	7746
	-1.0438e+04	-1.0466e+04	-1.0466e+04	-1.0466e+04	-1.0466e+04
	8.3622e-03	4.9100e-08	2.5557e-11	1.7486e-14	1.5548e-09
	0	0	0	0	0
	20	12681	5649	7076	13237
	00:00:02:10	00:02:51:13	00:00:50:25	01:14:32:37	00:14:01:50
G65 8000	5062	5008	5010	5026	–
	-6.1967e+03	-6.2055e+03	-6.2055e+03	-6.2051e+03	–
	8.9344e-03	4.0923e-13	2.4743e-11	1.7440e-14	–
	0	0	0	0	–
	16	15495	6613	9177	–
	00:00:00:36	00:03:37:10	00:00:52:36	02:17:39:55	–
G66 9000	5750	5764	5766	5782	–
	-7.0624e+03	-7.0772e+03	-7.0772e+03	-7.0768e+03	–
	9.4618e-03	6.4942e-13	2.3288e-12	1.4708e-14	–
	0	0	0	0	–
	18	21969	8405	11565	–
	00:00:00:31	00:04:51:33	00:01:14:52	03:07:36:48	–
G67 10000	6266	–	6224	6278	–
	-7.7367e+03	–	-7.7444e+03	-7.7441e+03	–
	9.9992e-03	–	1.6629e-14	1.6371e-08	–
	0	–	0	0	–
	16	–	10335	14231	–
	00:00:02:13	–	00:01:38:47	05:21:16:12	–



(a) $t = 10$ (154 sec) (b) $t = 100$ (2085 sec)

(c) $t = 1000$ (38'712 sec) (d) $t = 10'000$ (460'110 sec)

(e) original

Figure F.2. Phase retrieval SDP: Imaging. *Reconstruction of an $n = 640^2$ pixel image from Fourier ptychography data. We solve an $n \times n$ phase retrieval SDP via SketchyCGAL with rank parameter $R = 5$ and show the images obtained at iterations $t = 10, 10^2, 10^3, 10^4$. The last subfigure is the original. See subsection 7.4.3.*

Table 4

We solve SDP relaxations of QAP instances from QAPLIB using SketchyCGAL. We compute the relative gap and compare it with the values for the CSDP method [26] with clique size $k = \{2, 3, 4\}$ and the PATH method [116] reported in [26, Tab. 4]. Smaller is better. See subsection 7.5.

Dataset	Optimum	SketchyCGAL	CSDP2	CSDP3	CSDP4	PATH
chr12a	9552	0	34.5	6	0	42.7
chr12b	9742	0	38.9	25.4	11.9	38.1
chr12c	11156	0	5.8	2.3	2.3	18.6
chr15a	9896	0.4	2.1	2.1	2.1	52
chr15b	7990	0	26.3	34.5	29.2	158.6
chr15c	9504	0	0	0	0	63.3
chr18a	11098	1.5	69.8	0.2	0.2	76.3
chr18b	1534	8.7	8.9	22.9	29.5	99.3
chr20a	2192	6.6	122.5	76.1	43.8	95.4
chr20b	2298	0	62.9	9.3	9.3	82.2
chr20c	14142	17.1	173	100.1	111.5	88.9
chr22a	6156	0	17.2	7.6	3	38.3
chr22b	6194	4.7	7.3	2.3	1	40.4
chr25a	3796	21.8	107	49.2	25.2	69.9
esc16a	68	2.9	8.8	11.8	11.8	11.8
esc16b	292	0	0	0.7	0	2.7
esc16c	160	3.8	5	7.5	8.7	6.3
esc16d	16	12.5	12.5	50	25	75
esc16e	28	14.3	14.3	7.1	14.3	21.4
esc16g	26	0	7.7	0	15.4	15.4
esc16h	996	0	1.6	0	1.6	16.9
esc16i	14	0	0	0	0	57.1
esc16j	8	0	0	0	0	75
esc32a	130	52.3	115.4	124.6	113.8	93.8
esc32b	168	64.3	109.5	114.3	111.9	88.1
esc32c	642	2.2	12.8	15.9	13.7	7.8
esc32d	200	14	38	36	39	21
esc32e	2	0	0	0	0	600
esc32g	6	0	0	0	0	366.7
esc32h	438	5	24.7	26.9	22.8	18.3
esc64a	116	6.9	60.3	53.4	60.3	106.9
esc128	64	28.1	250	206.3	175	221.9
ste36a	9526	15.1	70.2	74.7	74.2	76.3
ste36b	15852	15.7	188.8	204.3	211.9	158.6
ste36c	8239110	9.7	66	62.8	63.7	83.2

Table 5

We run SketchyCGAL for (the first of) 10^6 iterations or 48 hours of runtime, for solving SDP relaxations of QAP instances from QAPLIB. We report the upper bound after rounding (subsection 7.5.3), the objective value, the feasibility gap, the number of iterations, the memory usage (in MB), and the cpu time ('hh:mm:ss'). See subsection 7.5.

Dataset	Optimum	Upper bnd.	Objective	Feas. gap	Iteration	Memory	Time
chr12a	9552	9552	9 576,06	5.02e−5	1000000	69	02 : 58 : 40
chr12b	9742	9742	9 759,43	2.43e−5	1000000	69	02 : 57 : 36
chr12c	11156	11156	11 021,43	2.05e−4	1000000	69	02 : 58 : 09
chr15a	9896	9936	9 519,70	1.91e−4	1000000	82	04 : 43 : 27
chr15b	7990	7990	7 469,99	1.72e−4	1000000	82	04 : 43 : 35
chr15c	9504	9504	9 517,70	3.70e−5	1000000	82	04 : 43 : 29
chr18a	11098	11262	10 700,38	2.39e−4	1000000	82	07 : 05 : 36
chr18b	1534	1668	1 534,30	8.08e−5	1000000	82	07 : 42 : 28
chr20a	2192	2336	2 183,67	1.11e−4	1000000	69	10 : 10 : 16
chr20b	2298	2298	2 289,02	9.43e−5	1000000	69	10 : 12 : 48
chr20c	14142	16554	13 316,80	1.89e−4	1000000	69	10 : 14 : 18
chr22a	6156	6156	6 152,78	1.06e−4	1000000	65	13 : 01 : 58
chr22b	6194	6486	6 198,40	9.91e−5	1000000	65	13 : 05 : 36
chr25a	3796	4624	3 692,54	2.06e−4	1000000	65	24 : 57 : 56
esc16a	68	70	60,36	5.36e−5	1000000	69	07 : 52 : 29
esc16b	292	292	287,64	9.63e−4	1000000	65	12 : 22 : 31
esc16c	160	166	145,01	5.59e−5	1000000	65	09 : 11 : 47
esc16d	16	18	13,00	6.37e−5	1000000	69	06 : 12 : 40
esc16e	28	32	25,41	6.89e−5	1000000	69	06 : 15 : 12
esc16g	26	26	22,46	7.05e−5	1000000	69	06 : 23 : 49
esc16h	996	996	975,41	2.06e−4	1000000	65	12 : 34 : 09
esc16i	14	14	11,37	8.25e−5	1000000	69	05 : 45 : 60
esc16j	8	8	7,11	8.49e−5	1000000	69	05 : 22 : 29
esc32a	130	198	99,60	3.65e−4	799533	208	48 : 00 : 00
esc32b	168	276	118,36	2.12e−4	634813	130	48 : 00 : 00
esc32c	642	656	610,85	7.33e−4	496050	65	48 : 00 : 00
esc32d	200	228	187,33	2.95e−4	609878	130	48 : 00 : 00
esc32e	2	2	1,90	4.67e−4	1000000	65	32 : 39 : 09
esc32g	6	6	5,83	4.48e−4	1000000	65	35 : 43 : 20
esc32h	438	460	417,78	3.79e−4	468726	65	48 : 00 : 00
esc64a	116	124	97,78	7.00e−3	163634	162	48 : 00 : 00
esc128	64	82	52,32	2.21e−2	30566	556	48 : 00 : 00
ste36a	9526	10966	8 992,22	9.72e−4	301900	130	48 : 00 : 00
ste36b	15852	18336	15 315,13	7.97e−4	301660	130	48 : 00 : 00
ste36c	8239110	9035686	7 980 802,09	8.86e−4	301558	130	48 : 00 : 00

Table 6

We solve SDP relaxations of QAP instances from TSPLIB using SketchyCGAL. We compute the relative gap and compare it with the values for the CSDP method [26] with clique size $k = \{2, 3, 4\}$ and the PATH method [116] reported in [26, Tab. 6]. Smaller is better.

Dataset	Optimum	CGAL	CSDP2	CSDP3	CSDP4	PATH
att48	10628	73.8	213	236.5	233.6	329.8
bayg29	1610	34.8	114.3	115.8	114.3	210.1
bays29	2020	55.7	107.6	118.3	115.4	164.8
berlin52	7542	52	175	127.2	127.2	280.6
bier127	118282	60.4	216.4	193.8	193.8	234.2
brazil58	25395	86.5	248	200.8	200.8	337
burma14	3323	10.8	24.6	28.4	32.3	95.5
ch130	6110	127.7	352.4	380.6	380.6	621.3
ch150	6528	121.1	346.9	318.2	318.2	689.3
dantzig42	699	73.5	193.1	174	174	82
eil101	629	70.6	227.3	235.3	235.3	437.7
eil51	426	57.3	203.6	205.4	205.5	244.4
eil76	538	65.4	282.9	183	183	328.2
fri26	937	32.7	91.6	39.4	39.4	41.6
gr120	6942	126.7	445.2	261.6	261.6	617.6
gr137	69853	147.5	264.6	220.3	220.3	38.9
gr17	2085	13.4	46.8	32.4	44.9	86.9
gr21	2707	27.2	94.5	69.7	66.3	185.7
gr24	1272	31.4	89.2	86.2	73.9	129.4
gr48	5046	59.7	210.2	187.4	187.4	270.4
gr96	55209	117.5	228.9	201.7	201.7	46
hk48	11461	43.9	222.4	207.7	207.7	281.6
kroA100	21282	125	469.6	469	469	720.2
kroA150	26524	184	411	467.4	467.4	945.8
kroB100	22141	156.8	411.9	313.6	313.6	624.2
kroB150	26130	151.9	417.3	353.7	353.7	844.7
kroC100	20749	169.5	507.4	445.1	445.1	763
kroD100	21294	118.5	504.2	349.8	349.8	654.4
kroE100	22068	152.3	489.5	346.3	346.3	684.2
lin105	14379	167.2	303.1	234.8	234.8	248.4
pr107	44303	190.5	181.5	207.9	207.9	41.6
pr124	59030	166.6	293.8	180.2	180.2	67.6
pr136	96772	144.5	325.5	164.7	164.7	196.6
pr144	58537	264.6	255	283.7	283.7	59.8
pr76	108159	81	192.2	194	194	39.4
rat99	1211	95.3	236.4	161.5	161.5	444.1
rd100	7910	88.8	438.4	375.3	375.3	506.5
st70	675	70.8	300.9	320	317.9	387.9
swiss42	1273	35.5	163.2	190.4	190.8	194
ulysses16	6859	11.1	23.6	20.2	23.2	82.7
ulysses22	7013	26.4	64.5	57	59.7	126.3

Table 7

We run SketchyCGAL for (the first of) 10^6 iterations or 48 hours of runtime, for solving SDP relaxations of QAP instances from TSPLIB. We report the upper bound after rounding (subsection 7.5.3), the objective value, the feasibility gap, the number of iterations, the memory usage (in MB), and the cpu time ('hh:mm:ss'). See subsection 7.5.

Dataset	Optimum	Upper bnd.	Objective	Feas. gap	Iteration	Memory	Time
att48	10628	18474	9 072,72	6.92e−4	311240	74	48 : 00 : 00
bayg29	1610	2170	1 498,60	5.40e−4	1000000	195	42 : 12 : 27
bays29	2020	3145	1 855,27	6.46e−4	1000000	195	42 : 19 : 21
berlin52	7542	11463	6 658,47	1.74e−3	266403	134	48 : 00 : 00
bier127	118282	189679	111 408,10	4.76e−2	23283	941	48 : 00 : 00
brazil58	25395	47362	18 336,86	1.70e−3	202400	129	48 : 00 : 00
burma14	3323	3682	3 153,63	5.14e−4	1000000	86	04 : 45 : 13
ch130	6110	13911	6 829,52	5.15e−2	20672	1077	48 : 00 : 00
ch150	6528	14432	11 021,38	1.39e−1	13925	1605	48 : 00 : 00
dantzig42	699	1213	589,35	5.11e−4	457221	129	48 : 00 : 00
eil101	629	1073	606,08	1.47e−2	46919	494	48 : 00 : 00
eil51	426	670	406,02	1.72e−3	286697	200	48 : 00 : 00
eil76	538	890	515,78	6.18e−3	103354	286	48 : 00 : 00
fri26	937	1243	858,28	6.57e−4	1000000	64	25 : 41 : 33
gr120	6942	15740	7 246,27	2.79e−2	27503	783	48 : 00 : 00
gr137	69853	172902	96 228,89	5.61e−2	19503	1206	48 : 00 : 00
gr17	2085	2365	1 800,49	3.83e−4	1000000	81	07 : 05 : 05
gr21	2707	3444	2 570,04	5.56e−4	1000000	64	12 : 12 : 17
gr24	1272	1672	1 136,65	3.53e−4	1000000	64	17 : 51 : 17
gr48	5046	8058	4 453,05	1.63e−3	357067	64	48 : 00 : 00
gr96	55209	120060	51 189,66	8.00e−3	58132	346	48 : 00 : 00
hk48	11461	16491	10 485,32	8.89e−4	357176	69	48 : 00 : 00
kroA100	21282	47891	19 953,20	8.72e−3	53552	462	48 : 00 : 00
kroA150	26524	75326	57 289,34	1.85e−1	11901	1605	48 : 00 : 00
kroB100	22141	56865	20 531,08	1.13e−2	43381	527	48 : 00 : 00
kroB150	26130	65832	57 292,86	2.02e−1	11722	1605	48 : 00 : 00
kroC100	20749	55920	19 831,30	1.08e−2	43352	509	48 : 00 : 00
kroD100	21294	46518	19 710,13	1.19e−2	43415	527	48 : 00 : 00
kroE100	22068	55680	20 463,81	1.06e−2	43352	557	48 : 00 : 00
lin105	14379	38417	12 640,70	1.13e−2	39662	440	48 : 00 : 00
pr107	44303	128712	36 333,09	1.32e−2	35009	651	48 : 00 : 00
pr124	59030	157381	74 152,15	3.76e−2	22049	900	48 : 00 : 00
pr136	96772	236593	135 053,09	6.06e−2	17446	1163	48 : 00 : 00
pr144	58537	213397	122 144,46	1.43e−1	14698	1383	48 : 00 : 00
pr76	108159	195718	91 294,27	2.50e−2	92634	287	48 : 00 : 00
rat99	1211	2365	1 206,81	9.76e−3	46602	462	48 : 00 : 00
rd100	7910	14935	7 459,54	9.89e−3	44863	527	48 : 00 : 00
st70	675	1153	586,71	2.92e−3	153971	134	48 : 00 : 00
swiss42	1273	1725	1 115,15	5.88e−4	599689	129	48 : 00 : 00
ulysses16	6859	7622	5 968,53	8.00e−4	1000000	73	05 : 16 : 03
ulysses22	7013	8865	5 999,82	1.03e−3	1000000	69	12 : 04 : 24

REFERENCES

[1] P.-A. Absil, C. Baker, and K. Gallivan, *Trust-region methods on Riemannian manifolds*, Found. Comput. Math., 7 (2007), pp. 303–330.

[2] A. Y. Alfakih, A. Khandani, and H. Wolkowicz, *Solving Euclidean distance matrix completion problems via semidefinite programming*, Comput. Optim. Appl., 12 (1999), pp. 13–30.

[3] F. Alizadeh, *Combinatorial optimization with interior point methods and semi-definite matrices*, PhD thesis, Univ. Minnesota, 1991.

[4] F. Alizadeh, *Interior point methods in semidefinite programming with applications to combinatorial optimization*, SIAM J. Optim., 5 (1995), pp. 13–51.

[5] F. Alizadeh, J.-P. A. Haeberly, and M. L. Overton, *Complementarity and nondegeneracy in semidefinite programming*, Math. Programming, 77 (1997), pp. 111–128.

[6] F. Alizadeh, J.-P. A. Haeberly, and M. L. Overton, *Primal-dual interior point methods for semidefinite programming: Convergence rates, stability and numerical results*, SIAM J. Optim., 8 (1998), pp. 746–768.

[7] Z. Allen-Zhu, Y. T. Lee, and L. Orecchia, *Using optimization to obtain a width-independent, parallel, simpler, and faster positive SDP solver*, 2015, https://arxiv.org/abs/1507.02259.

[8] Z. Allen-Zhu and Y. Li, *Follow the compressed leader: Faster online learning of eigenvectors and faster MMWU*, June 2017, https://arxiv.org/abs/1701.01722.

[9] M. F. Anjos and J. B. Lasserre, *Handbook on Semidefinite, Conic and Polynomial Optimization*, Springer USA, 2012.

[10] S. Arora, E. Hazan, and S. Kale, *Fast algorithms for approximate semidefinite programming using the multiplicative weights update method*, in Proc. 46th Ann. IEEE Symp. Foundations of Computer Science, FOCS '05, Washington, DC, USA, 2005, pp. 339–348.

[11] S. Arora and S. Kale, *A combinatorial, primal-dual approach to semidefinite programs*, J. ACM, 63 (2016), pp. 12:1–12:35.

[12] F. Bach, *Duality between subgradient and conditional gradient methods*, SIAM J. Optim., 25 (2015), pp. 115–129.

[13] D. Bader, H. Meyerhenke, P. Sanders, and D. Wagner, *Graph partitioning and graph clustering. 10th DIMACS Implementation Challenge Workshop,* February 2012, https://www.cise.ufl.edu/research/sparse/matrices/DIMACS10/index.html (accessed October 2019).

[14] M. Baes, M. Bürgisser, and A. Nemirovski, *A randomized mirror-prox method for solving structured large-scale matrix saddle-point problems*, SIAM J. Optim., 23 (2013), pp. 934–962.

[15] R. Balan, B. G. Bodmann, P. G. Casazza, and D. Edidin, *Painless reconstruction from magnitudes of frame coefficients*, J. Fourier Anal. Appl., 15 (2009), pp. 488–501.

[16] R. Balan, P. Casazza, and D. Edidin, *On signal reconstruction without phase*, Appl. Comp. Harmonic Anal., 20 (2006), pp. 345–356.

[17] A. S. Bandeira, *Convex relaxations for certain inverse problems on graphs*, PhD thesis, Princeton Univ., 2015.

[18] A. Barvinok, *A course in convexity*, AMS, Providence, RI, 2002.

[19] A. Ben-Tal and A. Nemirovski, *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications*, SIAM, Philadelphia, PA, 2001.

[20] D. P. Bertsekas, *Constrained optimization and Lagrange multiplier methods*, Computer Science and Applied Mathematics, Academic Press, New York, NY, 1982.

[21] S. Bhojanapalli, N. Boumal, P. Jain, and P. Netrapalli, *Smoothed analysis for low-rank solutions to semidefinite programs in quadratic penalty form*, in Proc. 31st Conf. Learning Theory, vol. 75, 2018, pp. 3243–3270.

[22] N. Boumal, B. Mishra, P.-A. Absil, and R. Sepulchre, *Manopt, a Matlab toolbox for optimization on manifolds*, J. Mach. Learn. Res., 15 (2014), pp. 1455–1459.

[23] N. Boumal, V. Voroninski, and A. Bandeira, *The non-convex Burer–Monteiro approach works on smooth semidefinite programs*, in Adv. Neural Information Processing Systems 29, 2016, pp. 2757–2765.

[24] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, *Distributed optimization and statistical learning via the alternating direction method of multipliers*, Found. Trends Mach. Learn., 3 (2011),

pp. 1–122.
[25] M. Brand, *Fast low-rank modifications of the thin singular value decomposition*, Linear Alg. Appl., 415 (2006), pp. 20 – 30.
[26] J. F. S. Bravo Ferreira, Y. Khoo, and A. Singer, *Semidefinite programming approach for the quadratic assignment problem with a sparse graph*, Comput. Optim. Appl., 69 (2018), pp. 677–712.
[27] S. Burer and R. D. Monteiro, *A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization*, Math. Prog., 95 (2003), pp. 329–357.
[28] S. Burer and R. D. Monteiro, *Local minima and convergence in low-rank semidefinite programming*, Math. Prog., 103 (2005), pp. 427–444.
[29] S. Burer and D. Vandenbussche, *Solving lift-and-project relaxations of binary integer programs*, SIAM J. Optim., 16 (2006), pp. 726–750.
[30] R. E. Burkard, S. E. Karisch, and F. Rendl, *QAPLIB – a quadratic assignment problem library*, J. Global Opt., 10 (1997), pp. 391–403.
[31] E. J. Candès, X. Li, and M. Soltanolkotabi, *Phase retrieval from coded diffraction patterns*, Appl. Comp. Harmonic Anal., 39 (2015), pp. 277 – 299.
[32] Y. Carmon, J. C. Duchi, S. Aaron, and T. Kevin, *A rank-1 sketch for matrix multiplicative weights*, in Proc. 32nd Conf. Learning Theory, vol. 99, 25–28 Jun 2019, pp. 589–623.
[33] A. Chai, M. Moscoso, and G. Papanicolaou, *Array imaging using intensity-only measurements*, Inv. Prob., 27 (2010), p. 015005.
[34] D. Cifuentes, *Burer–Monteiro guarantees for general semidefinite programs*, 2019, https://arxiv.org/abs/1904.07147.
[35] K. L. Clarkson, *Coresets, sparse greedy approximation, and the Frank–Wolfe algorithm*, ACM Trans. Algorithms, 6 (2010), pp. 63:1–63:30.
[36] J. L. Conrad, *Marburg virus hemorrhagic fever: Cytoarchitecture and histopathology*, https://pixnio.com/de/wissenschaft/mikroskopische-aufnahmen/hamorrhagisches-fieber-marburg-virus/cytoarchitectural-histopathologischen-erkannt-niere-probe-marburg-patient (accessed November 2019). Public domain (CC0 license).
[37] C. Delorme and S. Poljak, *Laplacian eigenvalues and the maximum cut problem*, Math. Prog., Ser. A, 62 (1993), pp. 557–574.
[38] C. Delorme and S. Poljak, *The performance of an eigenvalue bound on the max-cut problem in some classes of graphs*, Discrete Math., 111 (1993), pp. 145–156.
[39] L. Ding, A. Yurtsever, V. Cevher, J. A. Tropp, and M. Udell, *An optimal-storage approach to semidefinite programming using approximate complementarity*, 2019, https://arxiv.org/abs/1902.03373.
[40] M. Fazel, *Matrix Rank Minimization with Applications*, PhD thesis, Stanford University, Palo Alto, CA, USA, 2002.
[41] J. R. Fienup, *Phase retrieval algorithms: a comparison*, Appl. Optics, 21 (1982), pp. 2758–2769.
[42] M. Frank and P. Wolfe, *An algorithm for quadratic programming*, Naval Res. Logistics Quart., 3 (1956), pp. 95–110.
[43] D. Garber and E. Hazan, *Approximating semidefinite programs in sublinear time*, in Adv. Neural Information Processing Systems 25, 2011.
[44] D. Garber and E. Hazan, *Sublinear time algorithms for approximate semidefinite programming*, Math. Prog., 158 (2016), pp. 329–361.
[45] G. Gidel, F. Pedregosa, and S. Lacoste-Julien, *Frank–Wolfe splitting via augmented Lagrangian method*, in Proc. 21st Int. Conf. Artificial Intelligence and Statistics, vol. 84, 2018, pp. 1456–1465.
[46] A. Gittens, *Topics in Randomized Numerical Linear Algebra*, PhD thesis, California Institute of Technology, Pasadena, CA, USA, 2013.
[47] M. X. Goemans and D. P. Williamson, *Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming*, J. ACM, 42 (1995), pp. 1115–1145.
[48] G. H. Golub and C. F. Van Loan, *Matrix computations*, Johns Hopkins Univ. Press, Baltimore, MD, fourth ed., 2013.
[49] N. Halko, P. G. Martinsson, and J. A. Tropp, *Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Rev., 53 (2011), pp. 217–288.
[50] E. Hazan, *Sparse approximate solutions to semidefinite programs*, in LATIN 2008: Theoretical Infor-

matics, 2008, pp. 306–316.
[51] C. Helmberg, *Numerical evaluation of SBmethod*, Math. Prog., Ser. B, 95 (2003), pp. 381–406.
[52] C. Helmberg and F. Oustry, *Bundle methods to minimize the maximum eigenvalue function*, in Handbook of Semidefinite Programming, Springer, Boston, MA, 2000.
[53] C. Helmberg and F. Rendl, *A spectral bundle method for semidefinite programming*, SIAM J. Optim., 10 (2000), pp. 673–696.
[54] S. Homer and M. Peinado, *Design and performance of parallel and distributed approximation algorithms for maxcut*, J. Parallel Distrib. Comput., 46 (1997), pp. 48 – 61.
[55] R. Horstmeyer, R. Y. Chen, X. Ou, B. Ames, J. A. Tropp, and C. Yang, *Solving ptychography with a convex relaxation*, New J. Phys., 17 (2015), p. 053044.
[56] Q. Huang, Y. Chen, and L. Guibas, *Scalable semidefinite relaxation for maximum a posterior estimation*, in Proc. 31st Int. Conf. Machine Learning, vol. 32(2), 2014, pp. 64–72.
[57] M. Jaggi, *Revisiting Frank–Wolfe: Projection-free sparse convex optimization*, in Proc. 30th Int. Conf. Machine Learning, vol. 28(1), 2013, pp. 427–435.
[58] L. K. Jones, *A simple lemma on greedy approximation in Hilbert space and convergence rates for projection pursuit regression and neural network training*, Ann. Statist., 20 (1992), pp. 608–613.
[59] R. Jonker and A. Volgenant, *A shortest augmenting path algorithm for dense and sparse linear assignment problems*, Computing, 38 (1987), pp. 325–340.
[60] A. P. Kamath and N. K. Karmarkar, *A continuous method for computing bounds in integer quadratic optimization problems*, J. Global Optim., 2 (1991), pp. 229–241.
[61] A. P. Kamath and N. K. Karmarkar, *An $O(nL)$ iteration algorithm for computing bounds in quadratic optimization problems*, in Complexity in Numerical Optimization, World Scientific Publishing, July 1993, pp. 254–268.
[62] N. K. Karmarkar, *A new polynomial-time algorithm for linear programming*, Combinatorica, 4 (1984), pp. 373–395.
[63] R. M. Karp, *Reducibility among combinatorial problems*, in Complexity of computer computations, 1972, pp. 85–103.
[64] K. Krishnan and T. Terlaky, *Interior point and semidefinite approaches in combinatorial optimization*, in Graph Theory and Combinatorial Optimization, Springer, Boston, MA, 2005, ch. 1.
[65] J. Kuczyński and H. Woźniakowski, *Estimating the largest eigenvalue by the power and Lanczos algorithms with a random start*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 1094–1122.
[66] H. Kuhn, *The Hungarian Method for the assignment problem*, Naval Res. Logistics Quart., 2 (1955), pp. 83–97.
[67] B. Kulis, A. C. Surendran, and J. C. Platt, *Fast low-rank semidefinite programming for embedding and clustering*, in Proc. 11th Int. Conf. Artificial Intelligence and Statistics, vol. 2, San Juan, Puerto Rico, 21–24 Mar 2007, pp. 235–242.
[68] J. Lavaei and S. H. Low, *Zero duality gap in optimal power flow problem*, IEEE Trans. Power Sys., 27 (2012), pp. 92–107.
[69] Y. T. Lee and S. Padmanabhan, *An $\tilde{O}(m/\varepsilon^3.5)$-cost algorithm for semidefinite programs with diagonal constraints*. Available at http://arXiv.org/abs/1902.01859, March 2019.
[70] R. B. Lehoucq, D. C. Sorensen, and C. Yang, *ARPACK users' guide*, vol. 6 of Software, Environments, and Tools, SIAM, Philadelphia, PA, 1998. Solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods.
[71] E. S. Levitin and B. T. Poljak, *Minimization methods in the presence of constraints*, Ž. Vyčisl. Mat i Mat. Fiz., 6 (1966), pp. 787–823.
[72] H. Li, G. C. Linderman, A. Szlam, K. P. Stanton, Y. Kluger, and M. Tygert, *Algorithm 971: an implementation of a randomized algorithm for principal component analysis*, ACM Trans. Math. Software, 43 (2017), pp. Art. 28, 14.
[73] Y.-F. Liu, X. Liu, and S. Ma, *On the nonergodic convergence rate of an inexact augmented lagrangian framework for composite convex programming*, Math. Oper. Res., 44 (2019), pp. 632–650.
[74] E. M. Loiola, N. M. M. de Abreu, P. O. Boaventura-Netto, P. Hahn, and T. Querido, *A survey for the quadratic assignment problem*, Europ. J. Oper. Res., 176 (2007), pp. 657–690.
[75] A. Majumdar, G. Hall, and A. A. Ahmadi, *A survey of recent scalability improvements for semidefinite programming with applications to machine learning, control, and robotics*. Available at

http://arXiv.org/abs/1908.05209, Sep. 2019.

[76] M. Mesbahi and G. P. Papavassilopoulos, *On the rank minimization problem over a positive semidefinite linear matrix inequality*, IEEE Trans. Automatic Control, 42 (1997), pp. 239–243.

[77] L. Mirsky, *Symmetric gauge functions and unitarily invariant norms*, Quart. J. Math. Oxford Ser. (2), 11 (1960), pp. 50–59.

[78] MOSEK ApS, *The MOSEK optimization toolbox for MATLAB manual. Version 9.0.*, 2019.

[79] J. Munkres, *Algorithms for the assignment and transportation problems*, J. SIAM, 5 (1957), pp. 32–38.

[80] A. Nemirovski, *Prox-method with rate of convergence $O(1/t)$ for variational inequalities with Lipschitz continuous monotone operators and smooth convex-concave saddle point problems*, SIAM J. Optim., 15 (2004), pp. 229–251.

[81] Y. Nesterov, *Introductory Lectures on Convex Optimization: A Basic Course*, Springer US, 2004.

[82] Y. Nesterov, *Primal-dual subgradient methods for convex problems*, Math. Program., 120 (2009), pp. 221–259.

[83] Y. Nesterov and A. Nemirovski, *Self-concordant functions and polynomial time methods in convex programming*, Central Economic & Mathematic Institute report, USSR Acad. Sci., April 1990.

[84] Y. Nesterov and A. Nemirovski, *Interior-Point Polynomial Algorithms in Convex Programming*, SIAM, Philadelphia, PA, 1994.

[85] M. L. Overton and R. S. Womersley, *On the sum of the largest eigenvalues of a symmetric matrix*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 41–45, https://doi.org/10.1137/0613006.

[86] P. Parrilo, *Structured Semidefinite Programs and Semialgebraic Geometry Methods in Robustness and Optimization*, PhD thesis, California Institute of Technology, Pasadena, CA, 2000.

[87] G. Pataki, *On the rank of extreme matrices in semidefinite programs and the multiplicity of optimal eigenvalues*, Math. Oper. Res., 23 (1998), pp. 339–358.

[88] R. Peng and K. Tangwongsan, *Faster and simpler width-independent parallel algorithms for positive semidefinite programming*, in Proc. 24th Ann. ACM Symp. Parallelism in Algorithms and Architectures, 2012, pp. 101–108.

[89] T. Pumir, S. Jelassi, and N. Boumal, *Smoothed analysis of the low-rank approach for smooth semidefinite programs*, in Adv. Neural Information Processing Systems 31, 2018, pp. 2281–2290.

[90] A. Raghunathan, J. Steinhardt, and P. Liang, *Semidefinite relaxations for certifying robustness to adversarial examples*, in Adv. Neural Information Processing Systems 32, 2018, pp. 10900–10910.

[91] G. Reinelt, *TSPLIB*, http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/.

[92] D. M. Rosen, L. Carlone, A. S. Bandeira, and J. J. Leonard, *SE-sync: A certifiably correct algorithm for synchronization over the special euclidean group*, Int. J. Robot. Res., 38 (2019), pp. 95–125.

[93] M. F. Sahin, A. Eftekhari, A. Alacaoglu, F. Latorre, and V. Cevher, *An inexact augmented Lagrangian framework for nonconvex optimization with nonlinear constraints*, 2019, https://arxiv.org/abs/1906.11357.

[94] S. Sahni and T. Gonzalez, *P-complete approximation problems*, J. ACM, 23 (1976), pp. 555–565.

[95] A. Silveti-Falls, C. Molinari, and J. Fadili, *Generalized conditional gradient with augmented Lagrangian for composite minimization*, 2019, https://arxiv.org/abs/1901.01287.

[96] N. Srebro, *Learning with matrix factorizations*, PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 2004.

[97] J. Sturm, *Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones*, Optimization Methods and Software, 11–12 (1999), pp. 625–653.

[98] Y. Sun, Y. Guo, J. A. Tropp, and M. Udell, *Tensor random projection for low memory dimension reduction*, in NeurIPS Workshop on Relational Representation Learning, 2018.

[99] K. A. Toh, M. J. Todd, and R. H. Tütüncü, *SDPT3 — a Matlab software package for semidefinite programming, optimization methods and software*, Optim. Methods Software, 11 (1999), pp. 545–581.

[100] J. A. Tropp, A. Yurtsever, M. Udell, and V. Cevher, *Fixed-rank approximation of a positive-semidefinite matrix from streaming data*, in Adv. Neural Information Processing Systems 31, 2017, pp. 1225–1234.

[101] J. A. Tropp, A. Yurtsever, M. Udell, and V. Cevher, *Practical sketching algorithms for low-rank matrix approximation*, SIAM J. Matrix Anal. Appl., 38 (2017), pp. 1454–1485.

[102] J. A. Tropp, A. Yurtsever, M. Udell, and V. Cevher, *Streaming low-rank matrix approximation*

with an application to scientific simulation, SIAM J. Sci. Comput., 41 (2019), pp. A2430–A2463.

[103] K. Tsuda, G. Rätsch, and M. K. Warmuth, *Matrix exponentiated gradient updates for on-line learning and Bregman projections*, J. Mach. Learn. Res., 6 (2005), pp. 995–1018.

[104] I. Waldspurger, A. d'Aspremont, and S. Mallat, *Phase recovery, MaxCut and complex semidefinite programming*, Math. Prog., 149 (2015), pp. 47–81.

[105] I. Waldspurger and A. Waters, *Rank optimality for the Burer–Monteiro factorization*, 2018, https://arxiv.org/abs/1812.03046.

[106] Z. Wen, D. Goldfarb, S. Ma, and K. Scheinberg, *Row by row methods for semidefinite programming*, IEOR report, Columbia University, 2009.

[107] Z. Wen, D. Goldfarb, and W. Yin, *Alternating direction augmented Lagrangian methods for semidefinite programming*, Math. Program. Comp., 2 (2010), pp. 203–230.

[108] L. Yang, D. Sun, and K.-C. Toh, *SDPNAL+: A majorized semismooth Newton-CG augmented Lagrangian method for semidefinite programming with nonnegative constraints*, Math. Prog. Comput., 7 (2015), pp. 331–366.

[109] Y. Ye, *GSet random graphs*, https://www.cise.ufl.edu/research/sparse/matrices/Gset/ (accessed October 2019).

[110] A. Yurtsever, O. Fercoq, and V. Cevher, *A conditional-gradient-based augmented Lagrangian framework*, in Proc. 36th Int. Conf. Machine Learning, vol. 97, 09–15 Jun 2019, pp. 7272–7281.

[111] A. Yurtsever, O. Fercoq, F. Locatello, and V. Cevher, *A conditional gradient framework for composite convex minimization with applications to semidefinite programming*, in Proc. 35th Intl. Conf. Machine Learning, vol. 80, 2018, pp. 5727–5736.

[112] A. Yurtsever, Y.-P. Hsieh, and V. Cevher, *Scalable convex methods for phase retrieval*, in 6th Int. IEEE Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP), 2015, pp. 381–384.

[113] A. Yurtsever, Q. Tran Dinh, and V. Cevher, *A universal primal-dual convex optimization framework*, in Adv. Neural Information Processing Systems 28, 2015, pp. 3150–3158.

[114] A. Yurtsever, Q. Tran-Dinh, and V. Cevher, *Universal primal-dual proximal-gradient methods*, 2015, https://arxiv.org/abs/1502.03123v2.

[115] A. Yurtsever, M. Udell, J. Tropp, and V. Cevher, *Sketchy Decisions: Convex Low-Rank Matrix Optimization with Optimal Storage*, in Proc. 20th Int. Conf. Artificial Intelligence and Statistics, vol. 54, 20–22 Apr 2017, pp. 1188–1196.

[116] M. Zaslavskiy, F. Bach, and J. Vert, *A path following algorithm for the graph matching problem*, IEEE Trans. Pattern Anal. Mach. Intel., 31 (2009), pp. 2227–2242.

[117] Q. Zhao, S. E. Karisch, F. Rendl, and H. Wolkowicz, *Semidefinite programming relaxations for the quadratic assignment problem*, J. Comb. Optim., 2 (1998), pp. 71–109.

[118] X.-Y. Zhao, D. Sun, and K.-C. Toh, *A Newton-CG augmented Lagrangian method for semidefinite programming*, SIAM J. Optim., 20 (2010), pp. 1737–1765.

[119] G. Zheng, R. Horstmeyer, and C. Yang, *Wide-field, high-resolution Fourier ptychography microscopy*, Nat. Photonics, 7 (2013), pp. 739–745.